明　細　書

変換装置及び変換方法及び変換プログラム及び変換プログラムを記録したコンピュータ読み取り可能な記録媒体

技術分野

本発明は、プログラム変換装置及びその方法に関するものである。

背景技術

従来からバッチ処理を行うコボルプログラムをＣＯＲＢＡ、ＣＯＭなどの技術を用いて、ほぼ全体を人手によってクライアント／サーバ型のコボルプログラムに変換しようとする試みは存在した（例えば、非特許文献１（ＮＥＣ　Ｓｏｌｕｔｉｏｎｓ、　Ｏｐｅｎ　ＣＯＢＯＬ　Ｆａｃｔｏｒｙ　２１／　ＯｂｊｅｃｔＰａｒｔｎｅｒ　Ｐｒｏ、　[平成１４年７月１２日検索]、インターネット＜ＵＲＬ：http://www.sw.nec.co.jp/cced/ocf21/objptnpro/seihin.html＞）参照）。

図７８は一連のコボルプログラムをＣＯＲＢＡ、ＣＯＭなどの技術を用いて、クライアント／サーバ型コボルプログラムに変換する動作を示した従来図である。なお、ここで変換された結果は、クライアント／サーバ型ではあるが従来のコボルプログラムでありオブジェクト指向コボルプログラムではない。

このような試みでは、従来のコボル言語の手法を用いてコーディングされたコボルプログラム１００をクライアント／サーバ型のプログラムに変換する。まず、人間が一連のコボルプログラム１００

のソースコードに記載された内容を理解する。そして、人間によってコボルプログラム１００を内容的にまとまりのある個々のプログラムに分解する必要がある。また、ＣＯＲＢＡやＣＯＭなどと先程分解された個々のプログラムとを連携するためのインターフェースをインターフェース定義言語（ＩＤＬ：ＩＮＴＥＲＦＡＣＥ　ＤＥＦＩＮＩＴＩＯＮ　ＬＡＮＧＵＡＧＥ）によって記述する必要があるが、その作業にも人手が必要である。非特許文献１では、このＩＤＬ作成の一部のみを自動化している。

このように、コボルプログラム１００のソースコードから最終プログラム３００のソースコードを生成するために行なわれる変換処理は、従来においては自動化されていないか、自動化されている部分があっても手動で行なわなければならない部分があり、人間の労力を軽減する変換方法が望まれていた。すなわち、従来、メインフレーム機器４０００などに使われていたコボルプログラム１００の資源をクライアント／サーバシステム（分散システム）においても活用可能にするために、人的労力をかけない自動化された変換装置及び方法が望まれていた。

さらに、情報システムへの要請が集中型処理から分散型処理に変化している近年の状況を踏まえると、過去にコボルプログラムで構築した業務ロジックを再利用する方法がより一層求められていた。

本発明は、既存のプログラムを新しい技術のソフトウェアに適した構造に変換することを目的とする。

発明の開示

この発明に係る変換装置は、バッチ処理をするプログラムをソースコードの形式で記憶する記憶部と、

上記記憶部が記憶したプログラムのソースコードを１以上のまとまりある処理に区切り、区切った処理を節として節毎の役割を各節の意味情報として判断する節判断部と、

上記節判断部が判断した各節の意味情報に基づいて上記記憶部が記憶したプログラムのソースコードからソースコード変換のための変換情報を抽出し、抽出した変換情報に基づいてプログラムのソースコードをクライアント機器用の変換結果プログラムのソースコードとサーバ機器用の変換結果プログラムのソースコードとの２つからなる変換結果プログラムのソースコードに変換する抽出／変換部とを備える。

上記抽出／変換部は、上記２つの変換結果プログラムのソースコードを、オブジェクト指向プログラムのソースコードに変換することを特徴とする。

上記抽出／変換部は、所定のデータ構造と手続きをもつ複数のクラスに対応した複数のオブジェクト指向プログラムのテンプレートを生成し、上記２つの変換結果プログラムのソースコードから所定のデータ構造と手続きからなる情報を複数抽出し、抽出した各情報をテンプレートの対応する部分に適用することによって上記２つの変換結果プログラムのソースコードを複数のオブジェクト指向プログラムのソースコードに変換する。

この発明に係る変換装置は、バッチ処理をするプログラムをソースコードの形式で記憶する記憶部と、

上記記憶部が記憶したプログラムのソースコードを１以上のまと

まりある処理に区切り、区切った処理を節として節毎の役割を各節の意味情報として判断する節判断部と、

所定のデータ構造と手続きをもつ複数のクラスに対応した複数のオブジェクト指向プログラムのテンプレートを生成し、上記節判断部が判断した各節の意味情報に基づいて上記記憶部に記憶されたバッチ処理をするプログラムのソースコードから所定のデータ構造と手続きからなる情報を複数抽出し、抽出した各情報をテンプレートの対応する部分に適用することによって上記記憶部が記憶したプログラムのソースコードを複数のオブジェクト指向プログラムのソースコードに変換する。

上記変換装置は、さらに、

上記記憶部が記憶したプログラムのソースコードの役割をプログラムの意味情報として判断するプログラム判断部を備え、

上記抽出／変換部は、上記プログラム判断部が判断したプログラムの意味情報と上記節判断部が判断した各節の意味情報とに基づいてプログラムのソースコードからプログラムのソースコードを変換するための変換情報を抽出する。

上記変換装置は、さらに、

上記記憶部が記憶したプログラムを構文解析する構文解析部とを備え、

上記節判断部は、上記構文解析部によって構文解析されたプログラムに含まれる各節の意味情報を判断する。

上記変換装置は、バッチ処理をするコボルプログラムのソースコ

ードを変換する。

この発明に係る変換方法は、バッチ処理をするプログラムをソースコードの形式で記憶し、

上記記憶したプログラムのソースコードを１以上のまとまりある処理に区切り、区切った処理を節として節毎の役割を各節の意味情報として判断し、

上記判断した各節の意味情報に基づいて上記記憶したプログラムのソースコードからソースコード変換のための変換情報を抽出し、抽出した変換情報に基づいてプログラムのソースコードをクライアント機器用の変換結果プログラムのソースコードとサーバ機器用の変換結果プログラムのソースコードとの２つからなる変換結果プログラムのソースコードに変換する。

この発明に係る変換プログラムは、バッチ処理をするプログラムをソースコードの形式で記憶する処理と、

上記記憶したプログラムのソースコードを１以上のまとまりある処理に区切り、区切った処理を節として節毎の役割を各節の意味情報として判断する処理と、

上記判断した各節の意味情報に基づいて上記記憶したプログラムのソースコードからソースコード変換のための変換情報を抽出し、抽出した変換情報に基づいてプログラムのソースコードをクライアント機器用の変換結果プログラムのソースコードとサーバ機器用の変換結果プログラムのソースコードとの２つからなる変換結果プログラムのソースコードに変換する処理とをコンピュータに実行させる。

この発明に係るコンピュータに実行させるための変換プログラムを記録したコンピュータ読み取り可能な記録媒体は、バッチ処理をするプログラムをソースコードの形式で記憶する処理と、

上記記憶したプログラムのソースコードを１以上のまとまりある処理に区切り、区切った処理を節として節毎の役割を各節の意味情報として判断する処理と、

上記判断した各節の意味情報に基づいて上記記憶したプログラムのソースコードからソースコード変換のための変換情報を抽出し、抽出した変換情報に基づいてプログラムのソースコードをクライアント機器用の変換結果プログラムのソースコードとサーバ機器用の変換結果プログラムのソースコードとの２つからなる変換結果プログラムのソースコードに変換する処理とを備える。

この発明に係る変換装置は、オフライン一括処理を行う手続き型のプログラムソースコードを入力データとして、このプログラムソースコードをオンライン一件別処理のプログラムに変換する第１の変換部と、このオンライン一件別処理のプログラムをクライアント／サーバ環境で動作するＷｅｂプログラムに変換する第２の変換部とを備えたことを特徴とする。

上記第１の変換部は、オフライン一括処理を行うプログラムソースコードとデータの命名規則と手続きの組み方の規則を入力データとして、上記プログラムソースコードをオンライン一件別処理を行うクライアント側クラスとサーバ側クラスの２種類のクラス・プログラムに変換し、

上記第２の変換部は、この２種類のクラス・プログラムを入力し、オブジェクト指向プログラムのソースコードを生成することを特徴とする。

上記第２の変換部は、クライアント側クラスをモデルクラスとビュークラスとコントローラクラスとの３種類のクラス・プログラムに変換し、またサーバ側クラスをセッションクラスとエンティティクラスとの２種類のクラス・プログラムに変換することを特徴とする。

上記第１の変換部は、上記プログラムのソースコード内にあるデータの定義を参照し、マスタファイルの定義とトランザクションファイルの定義とを判定して、プログラムの役割とそのプログラム内の要素が一連の処理の中で果たす役割とを検出し、プログラムの役割とその要素が一連の処理の中で果たす役割を表すラベルを付与する意味付与の前処理を備えたことを特徴とする。

上記第１の変換部は変換１プログラムを実行し、上記第２の変換部は変換２プログラムを実行し、

オフライン一括処理を行うプログラムは種類分けされ、

上記変換１プログラムと上記変換２プログラムは、オフライン一括処理を行うプログラムの各種類ごとに対応して作成され、

上記第１の変換部と上記第２の変換部とは、入力したオフライン一括処理を行うプログラムの種類に対応して作成された変換１プログラムと変換２プログラムをそれぞれ実行することを特徴とする。

この発明に係る変換方法は、

オフライン一括処理を行う手続き型のプログラムソースコードを入力し、

このプログラムソースコードをオンライン一件別処理のプログラムに変換し、

このオンライン一件別処理のプログラムをクライアント／サーバ環境で動作するＷｅｂプログラムに変換することを特徴とする。

図面の簡単な説明

図１は、プログラム変換方法の一例を示す図である。

図２は、オフライン一括処理をオンライン一件別処理に変換する図である。

図３は、オフライン一括処理をオンライン一件別処理に変換する図である。

図４は、変換装置Ａ１０００の内部構成図である。

図５は、コボルプログラム１００から中間プログラム２００を取得する動作を示した図である。

図６は、コボルプログラム１００の一例を示す図である。

図７は、コボルプログラム１００を分割する流れ図である。

図８は、各プログラムの役割を判断する流れ図である。

図９は、入力チェックプログラム１０２の節の構造を示す図である。

図１０は、各節の役割を判断する流れ図である。

図１１は、マッチングプログラム１０７の節の構造を示す図である。

図１２は、各節の役割を判断する流れ図である。

図１３は、入力チェックプログラム１０２と結果出力プログラム１０９とインターフェースクラス２１０の対応図である。

図１４は、マッチングプログラム１０７とソートプログラム１０４とファイルクラス２２０の対応図である。

図１５は、プログラムの抽出、変換の詳細を示す図である。

図１６は、プログラムの抽出、変換の詳細を示す図である。

図１７は、入力チェックプログラム１０２の一部を示す図である。

図１８は、インターフェースクラス２１０の一部を示す図である。

図１９は、入力チェックプログラム１０２の一部を示す図である。

図２０は、インターフェースクラス２１０の一部を示す図である。

図２１は、入力チェックプログラム１０２を示す図である。

図２２は、インターフェースクラス２１０を示す図である。

図２３は、インターフェースクラス２１０を示す図である。

図２４は、結果出力プログラム１０９を示す図である。

図２５は、結果出力プログラム１０９を示す図である。

図２６は、プログラムの抽出、変換の詳細を示す図である。

図２７は、プログラムの抽出、変換の詳細を示す図である。

図２８は、プログラムの抽出、変換の詳細を示す図である。

図２９は、マッチングプログラム１０７を示す図である。

図３０は、マッチングプログラム１０７を示す図である。

図３１は、マッチングプログラム１０７を示す図である。

図３２は、ソートプログラム１０４を示す図である。

図３３は、ファイルクラス２２０のプログラムを示す図である。

図３４は、ファイルクラス２２０のプログラムを示す図である。

図３５は、変換装置Ｂ２０００の内部構成図である。

図３６は、オンライン一件別処理をよりオブジェクト指向的なオンライン一件別処理に変換する図である。

図３７は、インターフェースクラス２１０と3つのクラスの対応図である。

図３８は、プログラムの抽出、変換の詳細を示す図である。

図３９は、プログラムの抽出、変換の詳細を示す図である。

図４０は、プログラムの抽出、変換の詳細を示す図である。

図４１は、ビュークラス３１０のプログラムを示す図である。

図４２は、ビュークラス３１０のプログラムを示す図である。

図４３は、制御クラス３２０のプログラムを示す図である。

図４４は、モデルクラス３３０のプログラムを示す図である。

図４５は、モデルクラス３３０のプログラムを示す図である。

図４６は、ファイルクラス２２０と2つのクラスの対応図である。

図４７は、プログラムの抽出、変換の詳細を示す図である。

図４８は、プログラムの抽出、変換の詳細を示す図である。

図４９は、セッションクラス３４０のプログラムを示す図である。

図５０は、セッションクラス３４０のプログラムを示す図である。

図５１は、エンティティクラス３５０のプログラムを示す図である。

図５２は、エンティティクラス３５０のプログラムを示す図であ

る。

図５３は、エンティティクラス３５０のプログラムを示す図である。

図５４は、プログラム変換方法の他の一例を示す図である。

図５５は、変換装置Ｃ３０００の内部構成図である。

図５６は、変換装置Ａ、変換装置Ｂ、変換装置Ｃのコンピュータ基本構成図である。

図５７は、実施例の変換方法と従来の変換方法とを示す図である。

図５８は、実施例の、オフライン一括処理からオンライン一件別処理への処理方式の変換を示す図である。

図５９は、実施例の、サンプルを用いて検討した２段階の変換とプログラムの対応づけを示す図である。

図６０は、実施例の、クライアント側クラス２１０への対応づけを示す図である。

図６１は、実施例の、サーバ側クラス２２０への対応づけを示す図である。

図６２は、実施例の、Ｖｉｅｗ，　Ｃｏｎｔｒｏｌｌｅｒ，　Ｍｏｄｅｌクラス３１０、３２０、３３０への分解を示す図である。

図６３は、実施例の、Ｓｅｓｓｉｏｎ，　Ｅｎｔｉｔｙクラス３４０、３５０への分解を示す図である。

図６４は、実施例の、現行の集計・出力プログラム９０１の集計処理を示す図である。

図６５は、実施例の、一件別メソッドとして変換された変換後の集計・出力プログラム９０１ａの集計処理を示す図である。

図６６は、実施例の、現行の集計・出力プログラム９０１の集計

処理の手続きの流れを示す図である。

図６７は、実施例の、一件別メソッドとして変換された変換後の集計・出力プログラム９０１ａの集計処理の手続きの流れを示す図である。

図６８は、実施例の、集計・出力プログラム９０１のソースコードからのブレイク判定の除去による集計・出力プログラム９０１ａのソースコードへの変換を示す図である。

図６９は、実施例の、レコード形式変換プログラム９０２の変換の方針を示す図である。

図７０は、実施例の、現行のレコード形式変換プログラム９０２から変換後のレコード形式変換プログラム９０２ａへの手続きの変換を示す図である。

図７１は、実施例の、レコード形式変換プログラム９０２のソースコードからの繰返し指定の除去によるレコード形式変換プログラム９０２ａのソースコードへの変換を示す図である。

図７２は、実施例の、現行の出力指示データ作成プログラム９０３の処理を示す図である。

図７３は、実施例の、変換後の出力指示データ作成プログラム９０３ａの処理を示す図である。

図７４は、実施例の、現行の出力指示データ作成プログラム９０３から変換後の出力指示データ作成プログラム９０３ａへの手続きの変換を示す図である。

図７５は、実施例の、図６７の一件別メソッド（変換後の集計・出力プログラム９０１ａ）の集計処理のＣ／Ｓクラス２１０、２２０への分解を示す図である。

図７６は、実施例の、図７５のＣ／Ｓクラス２１０、２２０の集

計処理の５つのクラス３１０、３２０、３３０、３４０、３５０への分解を示す図である。

図７７は、実施例の、一括処理から一件別処理への変更で変化する行数の割合を示す図である。

図７８は、従来図である。

発明を実施するための最良の形態

実施の形態１.

本実施の形態では、既存の資産であるコボルプログラム１００のソースコードをまず中間プログラム２００のソースコードに変換し、さらに、最終プログラム３００のソースコードに変換する装置及び方法について説明する。

図１は、本実施の形態によるプログラム変換方法の一例を示す図である。コボルプログラム１００はメインフレーム機器における集中型処理及びオフライン処理及び一括処理（バッチ処理）の環境で動作するコボル言語で記述されたプログラムである。このプログラムのソースコードを変換装置Ａ１０００によって中間プログラム２００のソースコードに変換する。

中間プログラム２００はクライアント／サーバシステムである分散処理及びオンライン処理及び一件別処理の環境において動作するオブジェクト指向プログラムである。中間プログラム２００はインターフェースクラス２１０とファイルクラス２２０とから構成される。インターフェースクラス２１０はクライアント機器が処理するプログラムでありクライアント機器用の変換結果プログラムの一例である。ファイルクラス２２０はサーバ機器が処理するプログラムであり、サーバ機器用の変換結果プログラムの一例である。このよ

うに、コボルプログラム１００をインターフェースクラス２１０とファイルクラス２２０に分解することによって、メインフレーム機器上で動作していたコボルプログラム１００をネットワーク上につながれたクライアント機器とサーバ機器を連携させて処理するシステムに活用することができる。

なお、オブジェクトとは、外界の対象領域に存在するものについて、データ（属性）と手続き（メソッド）を一体化して表現したものをいう。また、クラスとは、オブジェクトの集合を抽象化した形で定義したものをいう。

中間プログラム２００のソースコードは、さらに、変換装置Ｂ２０００によって最終プログラム３００のソースコードに変換される。最終プログラム３００は、ＷＥＢやＶＢやＪａｖａ（登録商標）などに連携することが可能なオブジェクト指向プログラムの一例である。最終プログラム３００は、中間プログラム２００よりさらにオブジェクト指向性の高いプログラムである。また、最終プログラム３００は、ビュークラス３１０、制御クラス３２０、モデルクラス３３０、セッションクラス３４０、エンティティクラス３５０から構成されるプログラムである。

このように、既存のコボルプログラム１００をオブジェクト指向プログラムにまでソースコード変換することにより、インターネットなどのネットワーク上に構築された分散システム上で変換後のプログラムを動作させることができる。よって、既存のプログラムのさらなる有効活用が図れる。

まず、変換装置Ａ１０００が、コボルプログラム１００のソースコードを中間プログラム２００のソースコードへ変換する方法について説明する。

図２は、コボルプログラム１００によるオフライン一括処理を変換装置Ａ１０００によって中間プログラム２００によるオンライン一件別処理に変換する一例を示す図である。

左側の一括処理では、メインフレーム機器が複数の入力データをチェック、ソートし、トランザクションファイルを生成した後、旧マスタファイルとマッチング、更新して、新マスタファイルを出力するとともに、必要ならばエラーリストを出力する。変換装置Ａ１０００は、このように、コボルプログラム１００によるオフライン一括処理をクライアント／サーバシステムで行われるオンライン一件別処理に適合させるようにプログラムのソースコードを変換する。即ち、クライアント側では入力データの入力、チェックを行ない、トランザクションとしてサーバ機器に送る。サーバ機器ではクライアント機器から送られたトランザクションとマスタファイルをマッチングしてマスタファイルを更新するとともに、その結果をクライアント機器に伝える。

このように、変換装置Ａ１０００によってコボルプログラム１００のソースコードを中間プログラム２００のソースコードに変換することにより、オンラインの一件別処理が可能となる。

次に、上述したコボルプログラムでデータの追加や、更新削除などのトランザクションをオフライン一括処理を行うコボルプログラム１００を中間プログラム２００に変換することによってオンライン一件別処理を可能とする方法について、図３を用いてさらに詳細に説明する。

図３の左側はコボルプログラム１００によるオフライン一括処理を示し、右側は中間プログラム２００によるオンライン一件別処理を示す。

まず、入力・チェックプログラム１０２は、トランザクションファイル１０１のレコードを入力チェックし、チェック済みトランザクションファイル１０３を生成する。この処理の内容を記述した入力・チェックプログラム１０２のソースコードは、変換装置Ａ１００によって、右側のインターフェースクラス２１０のプログラムのソースコードに変換される。

左側の一括処理で生成されたチェック済みトランザクションファイル１０３のレコードは右側のトランザクションレコード２０３に対応するものである。

次に、左側の一括処理では、チェック済みトランザクションファイル１０３をソートプログラム１０４よってソートし、チェック・ソート済みトランザクションファイル１０５を生成する。このソート処理は、右側の処理には不要である。なぜなら、右側の処理はバッチ処理ではなく一件別処理であるため、トランザクションレコード２０３は直接マッチング処理の対象レコードとなるからである。

次に、左側の一括処理では、チェック・ソート済みトランザクションファイル１０５と旧マスタファイル１０６をマッチングプログラム１０７によりマッチングする。その結果、新マスタファイル１０８を得る。この処理に対応して右側の一件別処理では、トランザクションレコード２０３とマスタファイル２０６をマッチングする（２０７）処理が行われる。このマッチング処理はファイルクラス２２０に記述され、サーバ機器によって実行される。

最後に、左側の一括処理では、結果出力プログラム１０９がマッチング処理から得られた結果を出力する。この処理に対応して右側の一件別処理では、サーバ機器に結果が出力される（２０８）。

次に、上記左側の一括処理を行うプログラムから右側の一件別処

理を行うプログラムへ、そのソースコードを変換する変換装置Ａ１０００について説明する。

図４は、コボルプログラム１００のソースコードを中間プログラム２００のソースコードに変換する変換装置Ａ１０００の内部構成及び動作を示す図である。

変換装置Ａ１０００は変換前のコボルプログラム１００を入力する入力部１１００と、入力部１１００が入力したプログラムを複数のプログラムに分割する分割部１２００と、分割された各プログラムを構文解析する構文解析部１３００と、構文解析したプログラムから各プログラムの内容を判断するプログラム判断部１４００と、プログラム判断部１４００が判断した各プログラム中の複数の節の内容を判断する節判断部１５００と、節判断部１５００が判断した節を用いてコボルプログラム１００から中間プログラム２００へ変換するために必要なデータを抽出し、抽出したデータを用いて中間プログラム２００に変換する抽出／変換部１６００と、変換した中間プログラム２００を出力する出力部１７００と、入力部１１００が入力したプログラム等を記憶する記憶部１８００とから構成される。なお、記憶部１８００は必ずしも変換装置Ａ１０００の内部に存在する必要はなく、外部記憶装置を利用してもよい。

次に、図４に示された各内部構成を用いてコボルプログラム１００のソースコードを中間プログラム２００のソースコードに変換する動作を説明する。

図５は、左から右に進行し、左から入力されたコボルプログラム１００を変換して右の中間プログラム２００を取得する動作を示した図である。

前述した通り、コボルプログラム１００は、トランザクションフ

ファイル１０１を入力し、その内容をチェックする入力チェックプログラム１０２と、各チェック済みトランザクションレコードをソートするソートプログラム１０４と、チェック・ソート済みトランザクションファイル１０５と旧マスタファイル１０６をマッチングするマッチングプログラム１０７とマッチングした結果を出力する結果出力プログラム１０９とから構成されている。ただし、入力チェックプログラム１０２、ソートプログラム１０４、マッチングプログラム１０７、結果出力プログラム１０９のうちどれか一つが欠けていても構わない。

入力部１１００は、このコボルプログラム１００を入力する。

次に、分割部１２００は、４つのプログラムが１つにまとまった入力プログラムをそれぞれ４つのプログラムに分割する（Ｓ１２００）。ただし、コボルプログラム１００が一つのまとまったプログラムである場合には、分割部１２００は何も処理しない。

次に、構文解析部１３００は、分割された４つのプログラムに記載された各構文を解析する（Ｓ１３００）。

次に、プログラム判断部１４００は、構文解析部１３００によって行なわれた構文解析に基づいて各プログラムの役割を判断する（Ｓ１４００）。この判断の結果、入力チェックプログラム１０２は入力データの入力とチェック、ソートプログラム１０４は複数のレコードのソート、マッチングプログラム１０７はマスタファイルとのマッチング、結果出力プログラム１０９はマッチング結果の出力という役割を果たしていることが判断される。

次に、節判断部１５００がプログラム判断部１４００によって判断された各プログラム中の節の役割を判断する（Ｓ１５００）。ここで、節とはプログラムを一又は複数のまとまりある処理に区切り

、その区切った各処理をいう。

抽出／変換部１６００は、節判断部１５００によって判断された各節の役割からコボルプログラム１００のソースコードを中間プログラム２００のソースコードに変換するために必要なデータをコボルプログラム１００のソースコードから抽出し、抽出したデータを中間プログラム２００に適合するように変換する。

その結果、インターフェースクラス２１０とファイルクラス２２０とから構成される中間プログラム２００が生成され、出力部１７００によって出力される。

次に、変換装置Ａ１０００の各部の動作について説明する。

まず分割部１２００の動作について説明する。

図６は、入力部１１００が入力したコボルプログラム１００の一例である。ここでは、前述したように、コボルプログラム１００は４つのまとまりあるプログラムから構成されている。各プログラムの最終行には「ＥＮＤ　ＰＲＯＧＲＡＭ．」が存在する。

図７は、図６で示したコボルプログラム１００を複数のまとまりあるプログラムに分割する流れ図である。この流れ図は、ＰＡＤ（ＰＲＯＢＬＥＭ　ＡＮＡＬＹＳＩＳ　ＤＩＡＧＲＡＭＳ）によって記載されている。

ここでは、図６に示したプログラムに記述された見出し「ＩＤＥＮＴＩＦＩＣＡＴＩＯＮ　ＤＩＶＩＳＩＯＮ．」から次の見出し「ＥＮＤ　ＰＲＯＧＲＡＭ．」までを一つのプログラムと判断し、それぞれ別個の出力ファイルに格納する。

すなわち、まず新しい出力ファイルを開き（Ｓ１２０１）、プログラムが終わるまで以下の処理を繰り返す（Ｓ１２０２）。

プログラムを一行読み（Ｓ１２０３）、「ＥＮＤ　ＰＲＯＧＲＡ

M.」があるかどうかを判断する（Ｓ１２０４）。

「ＥＮＤ　ＰＲＯＧＲＡＭ.」が存在しない場合には、読み込んだ１行を出力ファイルに書き出す（Ｓ１２０７）。

「ＥＮＤ　ＰＲＯＧＲＡＭ.」が存在する場合には、まとまるある１つのプログラムの最終と判断し、出力ファイルを閉じ（Ｓ１２０５）、新しい出力ファイルを開く（Ｓ１２０６）。

この処理を繰り返すことによって、たとえば、図６に示されたコボルプログラム１００は４つのプログラム（入力・チェックプログラム１０２、ソートプログラム１０４、マッチングプログラム１０７、結果出力プログラム１０９）に分割される。

なお、図７においては見出し「ＥＮＤ　ＰＲＯＧＲＡＭ.」が存在するかを判断してプログラムを分割したが、入力ファイル名自体を用いて各々のファイル名から各まとまりあるプログラムに分割することも可能である。

また、分割部１２００は必ずしも設ける必要はなく、構文解析部１３００が直接入力されたプログラムに基づいて構文解析してもよい。

次に、分割部１２００によって分割された４個のプログラムの構文解析について説明する。構文解析部１３００は、分割された個々のプログラムを別個に構文解析する。その結果、各プログラムについてプログラムの階層構造を表現した構文解析木が得られる。

この構文解析木の各ノードには、プログラム中の命令に対する構文上の意味情報が付与されている。この構文解析木を作る操作はプログラム判断部１４００によって行なわれる。ここで、ノードとは、命令文の始まり及び終わりを境とし、プログラム中で意味を持つ最小の単位となる一連の語をまとめて格納したものをいう。

プログラム判断部１４００では各プログラム名の命名規則をもとに、入力チェックやマッチングなど、各プログラムの役割を判断し、判断した役割を意味情報として各プログラムに付与する。

ここで、プログラム名の命名規則について説明する。

プログラム中のプログラム名、ファイル、及びデータ項目については、予め各社内毎のコーディング規約によって命名規則が定められている。プログラム名については一連の処理に共通する名前と、入力、ソート、更新、結果出力の内どれかが識別できる名前が付いている。また、ファイル名にも各ファイルの役割を識別できる接辞が付与されている。

また、データ項目名には、ファイルのデータ項目ならばファイル名と同様にデータ項目の役割を識別できる接辞が付与されている。

これらプログラム中の命名規則に基づいて各プログラムに意味情報を付与することが可能である。

プログラム判断部１４００がプログラム名から各プログラムの役割を判断する流れ図を図８に示す。このフローは、プログラム判断部１４００によって実行される。

まず、プログラム判断部１４００は、構文解析部１３００によって解析された各プログラムについて（Ｓ１４０１）、プログラム名を抽出する（Ｓ１４０２）。

プログラム名が入力チェックならば、プログラムに意味情報として「入力チェック」を付与する（Ｓ１４０４）。

プログラム名がソートならばプログラムに意味情報「ソート」を付与する（Ｓ１４０５）。

プログラム名がマッチング更新であるならば、プログラムに意味情報「マッチング更新」を付与する（Ｓ１４０６）。

プログラム名が結果出力ならばプログラムに意味情報「結果出力」を付与する（Ｓ１４０７）。

次に、節判断部１５００が各プログラムの構文解析木の各節のノードを１つずつ抽出し、そのノードへ意味情報を付与する処理について説明する。

各プログラムは、前述したように、プログラム中のまとまりある処理である節から構成されている。

図９は、入力チェックプログラム１０２の各節の構造を示している。四角で囲まれた各処理が節に相当する。図９では、左の節が右の節を呼び出す構造になっている。

主処理（Ｓ１２９）はトランザクション入力処理（Ｓ１３８）と変換コントロール処理（Ｓ１３５）を呼び出している。トランザクション入力処理は、トランザクションファイル１０１からレコードを読み込む処理であり、変換コントロール処理は変換コントロール処理が呼び出す各処理（変換処理、トランザクション入力処理）を繰り返す制御処理である。

変換コントロール処理（Ｓ１３５）で行なわれる繰り返し処理は、変換処理（Ｓ１３９）とトランザクション入力処理（Ｓ１３８）を呼び出すことにより行なわれる。変換処理が行なわれた後には、チェック済トランザクション出力処理（Ｓ１４１）の節によってチェック済トランザクションレコードをファイルに書き出す処理を行う。なお、Ｓ１３９の変換処理では、トランザクションレコードをチェックし、正しいレコードを出力側に転記する処理を行なっている。

このようにして、節判断部１５００は、図９で示す入力・チェックプログラム１０２の節の構造から各節の役割を判断する。

節判断部１５００が図９に示した各節の役割を判断する流れ図を図１０に示す。

まず、節判断部１５００は節のノードを一つずつ抽出し（Ｓ１５０１）、ＯＰＥＮ文又はＲＥＡＤ文又はＷＲＩＴＥ文を含むかを判断する（Ｓ１５０２）。

節判断部１５００は、節に対応する構文解析木のノードにＯＰＥＮ文を含むと判断されている場合には節の役割として意味情報「主処理」を付与する（Ｓ１５０３）。ＲＥＡＤ文を含む場合には節の役割として意味情報「トランザクション入力」を付与する（Ｓ１５０４）。ＷＲＩＴＥ文を含む場合には節の役割の意味情報として「チェック済トランザクション出力」を付与する（Ｓ１５０５）。

次に、節判断部１５００は、他の節に対してチェック済出力をＰＥＲＦＯＲＭ文で読み出しているかを判断し（Ｓ１５０７）、呼び出している場合にはその節の役割として意味情報「変換処理」を付与する（Ｓ１５０８）。

節判断部１５００は、他の節に対して変換処理をＰＥＲＦＯＲＭ文で呼び出している判断される場合には（Ｓ１５１０）、その節の役割として意味情報「変換処理コントロール」を付与する（Ｓ１５１１）。このようにして、各節の役割として意味情報を自動的に付与することができる。

次に、節判断部１５００がマッチングプログラム１０７に対して節の判断と意味情報の付与を行う動作を説明する。

図１１は、マッチングプログラム１０７の節の構造を示している。図９と同様に四角で囲まれた処理は節を表し、左の処理が右の処理を呼び出す関係となっている。主処理（Ｓ１５７）はトランザクションを入力し（Ｓ１６５）、旧マスタファイルを入力し（Ｓ１６

６）、これらのデータを元に更新コントロールに基づいて更新処理を繰り返す。すなわち、更新コントロールでは、トランザクションキーとマスタキーを照合する合致処理を行ない（Ｓ１５９）、また次のマスタレコードを準備するマスタ処理（Ｓ１６９）と次のトランザクションレコードを準備するトランザクション処理（Ｓ１６０）に基づいて合致処理（Ｓ１５９）を繰り返す。その結果は新マスタファイルに出力される。このようにマッチングプログラム１０７は図１１に示す節の構造を持っているので、節判断部１５００はこれらの節の構造から各節の役割を判断する。

節判断部１５００が図１０に示した各節の役割を判断する流れ図を図１２に示す。

まず、節判断部１５００は、節が対応する構文解析木のノードを一つずつ抽出し（Ｓ１５２１）、ＯＰＥＮ文、トランザクションのＲＥＡＤ文、旧マスタファイルのＲＥＡＤ文、新マスタファイルのＲＥＡＤ文があるかどうかを判断する（Ｓ１５２２）。ＯＰＥＮ文がある場合には、その節の役割として、意味情報「主処理」を付与する（Ｓ１５２３）。トランザクションのＲＥＡＤ文がある場合にはその節に意味情報「トランザクション入力」を付与する（Ｓ１５２４）。旧マスタファイルのＲＥＡＤ文がある場合には、その節に意味情報「旧マスタファイル入力」を付与する（Ｓ１５２５）。新マスタファイルのＲＥＡＤ文がある場合にはその節に意味情報「新マスタファイル出力」を付与する（Ｓ１５２６）。

また、節判断部１５００は、残りの節について、前述したプログラム中の命名規則に従ってプログラム名、ファイル名、データ項目名を用いてどのレコード定義がトランザクションファイル、新マスタファイル、旧マスタファイルのいずれかのレコードかを判断する

ことができる（Ｓ１５２８）。

以上、節判断部１５００が入力・チェックプログラム１０２とマッチングプログラム１０７について節の判断を行なった後、抽出／変換部１６００が各節の判断に基づいて中間プログラム２００を生成するのに必要なデータを抽出し、変換する。次に、抽出／変換部１６００で行なわれる抽出／変換処理について説明する。

まず、前処理として中間プログラム２００に予め各クラス中のメソッドを作り込む作業が必要である。ここで、メソッドとは具体的処理方法（手段）をいう。

抽出／変換部１６００は、この前処理として中間プログラム２００を構成するインターフェースクラス２１０及びファイルクラス２２０の２つ構造を作り、各クラスにファイル名情報をつけておく。

また、上記２つのクラスには各々次の段落に示すメソッドを作り、中身は空にしておく。各メソッドには意味情報を付与しておき、後で抽出及び変換する際に各メソッドを特定できるようにする。

図１３は、入力チェックプログラム１０２と結果出力プログラム１０９とインターフェースクラス２１０の対応図である。

図１４は、マッチングプログラム１０７とソートプログラム１０４とファイルクラス２２０の対応図である。

図１３に示すように、抽出／変換部１６００は予め、インターフェースクラス２１０に画面表示入力メソッド（メソッド：ｄｉｓｐｌａｙＳｃｒｅｅｎ）とＵＩメインメソッド（メソッド：ｕｉＭａｉｎ）と入力チェックメソッド（メソッド：ｃｈａｎｇｅＭｏｄｅｌ）を作っておく。

また、抽出／変換部１６００は予め、ファイルクラス２２０にマッチング更新メソッド（メソッド：ｕｐｄａｔｅＲｅｃｏｒｄ）を

作っておく。

このような前処理を行なった後、図１３に示すように、抽出／変換部１６００は、入力チェックプログラム１０２と結果出力プログラム１０９からデータを抽出、変換し、クライアント機器５０００用の変換結果プログラムとしてインターフェースクラス２１０を生成する抽出変換処理を行う。また、図１４に示すように、抽出／変換部１６００はマッチングプログラム１０７とソートプログラム１０４からデータを抽出、変換し、サーバ機器６０００用の変換結果プログラムとしてファイルクラス２２０を生成する抽出変換処理を行う。

以下に、上記抽出／変換部１６００が行う抽出変換処理を説明する。

・入力チェックプログラム１０２と結果出力プログラム１０９からインターフェースクラス２１０への抽出変換処理

図１３に示すように、抽出／変換部１６００では入力チェックプログラム１０２と結果出力プログラム１０９とからインターフェースクラス２１０を生成するために対応付けを行う。

抽出／変換部１６００は入力チェックプログラム１０２から入力チェックのロジックを抽出し、インターフェースクラス２１０に対応付ける。また、抽出／変換部１６００は結果出力プログラム１０９から伝票定義を抽出し、インターフェースクラス２１０に対応付ける。

抽出／変換部１６００は、次に示す３つのロジックの対応付けに基づいて抽出変換処理を行う。

第１に入力チェックプログラム１０２では、トランザクションファイル１０１からレコードを読み込むロジックが存在していたが、

インターフェースクラス２１０では画面からデータを入力してレコード形式に格納するロジックに変換する必要がある。すなわち、トランザクションファイル１０１からレコードを読み込むロジックは不要となる。したがって、第１の対応付けから入力チェックプログラム１０２に存在していたトランザクションファイル１０１からレコードを読み込むロジックを無視する。

第２に入力チェックプログラム１０２では読み込んだ各レコードのデータをチェックするロジックが存在するが、インターフェースクラス２１０においてもこのロジックを継承する必要がある。

第３に入力チェックプログラム１０２ではデータが正しければレコードをチェック済トランザクションファイル１０３に書き出すロジックが存在していたが、インターフェースクラス２１０においてはデータが正しければレコードをファイルクラス２２０に送るロジックに変換する必要がある。すなわち、データが正しければレコードをチェック済トランザクションファイル１０３に書き出すロジックは不要となる。したがって、第３の対応付けから入力チェックプログラム１０２に存在していたチェック済トランザクションファイル１０３の定義部分とチェック済トランザクションファイル１０３に書き出すロジックを無視し、レコードをチェックして正しければファイルクラス２２０のメソッドに送るようにするロジックを加える。

また、一括処理から一件別処理へ処理方法が変更されるので、入力チェックプログラム１０２に存在していた次のレコードを準備する繰り返しのロジックを無視する。

これらの対応付けに基づいたプログラムの抽出、変換の詳細を図１５及び図１６に示す。図１５は入力チェックプログラム１０２か

らインターフェースクラス２１０のプログラムへの抽出及び変換の詳細を説明する図である。図１６は結果出力プログラム１０９からインターフェースクラス２１０のプログラムへの抽出及び変換を説明する図である。図１５及び図１６に示す位置番号は図１３及び図１４に示す対応付けの番号に対応している。すなわち、入力チェックプログラム１０２の見出し部にかかれたプログラム名からインターフェースクラス２１０の見出し部のクラス名及び環境部のファイルクラス２２０へのリポジトリ指定及びクラス終わり見出しの抽出変換を行うが、その詳細については図１５の位置番号（１－１）に示されている。

この抽出変換の詳細について、図１７と図１８を用いて説明する。

図１７は、入力チェックプログラム１０２の一部を示す図である。図１８は、インターフェースクラス２１０の一部を示す図である。

図１７の位置番号（１－１）で示されたステップから図１８の（１－１）で示されたステップが抽出変換される。

さらに、具体的に説明すると、図１７の見出し部（Ｓ１２１）中のＰＲＯＧＲＡＭ－ＩＤ（Ｓ１２２）の行から「在庫マスタ修正－入力チェック」を抽出し、図１８のインターフェースクラス２１０中の見出し部（Ｓ２２１）のクラスＩＤ「在庫マスタ修正ＵＩ」として変換する。

また、インターフェースクラス２１０内の環境部（Ｓ２２２）に記載されたファイルクラス２２０に対するリポジトリ指定の内部名として、入力チェックプログラム１０２から抽出した「在庫マスタ修正」にファイルクラス名の接辞を付加するとともに、外部名につ

いては各会社毎のコーディング規約中の命名規則に従ってファイルクラス２２０のファイル名を挿入する。

クラスの終わりを示す見出しについては後述する。

次に、図１７で示す入力チェックプログラム１０２のデータ部（Ｓ１２３）のトランザクションファイル定義に基づき、インターフェースクラス２１０のクラス変数を抽出変換する。具体的には図１７に示された位置番号（１－２）から抽出し、図１８のデータ部（Ｓ２２３）に示された位置番号（１－２）に変換する。このように入力チェックプログラム１０２のトランザクションファイルの接辞がついたレコードをインターフェースクラス２１０のクラス変数として記載し、画面からの入力をレコードの形式にしてファイルクラス２２０に送ることが可能となる。

次に、入力・チェックプログラム１０２中の手続き部からインターフェースクラス２１０の入力チェックメソッドを作成するための抽出変換処理について説明する。

図１９は、入力チェックプログラム１０２の手続き部を示している。図２０は、インターフェースクラス２１０の入力チェックメソッドの手続部とクラスの終わりの見出し部分を示している。

図１３に示すように、入力チェックプログラム１０２の手続き部（Ｓ１２８）はインターフェースクラス２１０の入力チェックメソッド「ｃｈａｎｇｅＭｏｄｅｌ」（Ｓ２２７）の手続き部に変換される。各変換処理についての詳細は、図１５の位置番号（１－４）から（１－８）に示されているが、この変換処理について図１９と図２０を用いて実際の具体的抽出変換を説明する。

図１９の手続き部（Ｓ１２８）中の主処理（Ｓ１２９）に記載された主処理節（Ｓ１３０からＳ１３４）を抽出し、この抽出した要

素から図２０に示す入力チェックメソッドの手続き部の主処理（Ｓ２２８）の処理内容を生成する変換処理を説明する。なお、抽出時の主処理部分の特定の仕方については前述したとおりであるので、省略する。

実際の構造及び構文上の変換については、まず、図１９で示すファイルのＯＰＥＮ文（Ｓ１３０）及びファイルのＣＬＯＳＥ文（Ｓ１３３）を無視する。また、変換処理コントロールへのＰＥＲＦＯＲＭ文（Ｓ１３２）から繰り返しの指定（ＵＮＴＩＬ指定）を無視する。ファイルのＯＰＥＮ文及びＣＬＯＳＥ文を無視するのは、インターフェースクラス２１０では入力は画面で行ない、チェック済レコードはファイルクラス２２０に出力するために不要となるためである。また、繰り返しの指定を無視するのは、中間プログラムではレコードを１件しか処理しないためである（一件別処理）。

次に、ＳＴＯＰ　ＲＵＮ．（Ｓ１３４）を中間プログラムの構文上必要なＥＸＩＴ　ＭＥＴＨＯＤ．に変換する。これは図２０のＳ２３８に示されている。

次に、位置番号（１－５）に示される処理の変換について説明する。

図１９において、変換処理コントロール（Ｓ１３５）に記載された具体的内容（Ｓ１３６、Ｓ１３７）を抽出し、トランザクション入力へのＰＥＲＦＯＲＭ文（Ｓ１３７）を消去する。これは、中間プログラムではレコードを１件しか処理しないので、次のトランザクションレコードを入力する必要がないためである。その結果、図２０にはＳ２３１及びＳ２３２のみが抽出される。

次に、位置番号（１－６）の抽出変換について説明する。

図１９のＳ１３８で示すトランザクションファイルのＲＥＡＤ文

を無視し、処理を何も行わないことを表すＣＯＮＴＩＮＵＥ文に置き換える。中間プログラムでは画面からの入力データが直ちにトランザクションレコードに入るため、ＲＥＡＤ文は不要となるためである。その結果、トランザクション入力処理は図２０のＳ２３４で示す部分のように変換される。

次に位置番号（１－７）の変換処理についての抽出変換を説明する。

図１９のＳ１３９で示す変換処理の内、Ｓ１４２のＭＯＶＥ文、すなわち、トランザクションレコードからチェック済トランザクションレコードへレコードを移動させるための文を無視する。中間プログラムによれば、トランザクションレコードをチェックしてデータが正しければそのままファイルクラス２２０にそのデータを送るためである。その結果、変換処理には図２０のＳ２３５で示す部分のみが抽出される。

次に、位置番号（１－８）の抽出変換処理について説明する。

ここでは、図１９のＳ１４１に示すトランザクション出力処理の抽出変換処理が行なわれる。具体的にはチェック済トランザクションファイルへのＷＲＩＴＥ文（Ｓ１４３）をファイルクラス２２０のマッチング更新メソッドへのＩＮＶＯＫＥ文に置き換える。中間処理プログラムにおいては、チェックしたレコードはファイルクラス２２０のマッチング更新メソッドへの引数となるからである。このようにして、抽出変換した結果を図２０のＳ２３６に示す。

なお、Ｓ２３７は位置番号（１－１）に対応してクラスの終わりの見出しを抽出変換した変換後のステップを示している。

以上のようにして抽出／変換部１６００はコボルプログラム１００に記載された入力チェックプログラム１０２のソースコードから

中間プログラム２００を構成するインターフェースクラス２１０のソースコードを自動的に抽出変換する。

実際の入力チェックプログラム１０２のソースコードを図２１に示す。また、入力チェックプログラム１０２のソースコードから抽出／変換部１６００によって抽出、変換されたインターフェースクラス２１０のソースコードを図２２、図２３に示す。すなわち、図２２の第１行であるステップ１（０００００１）から図２３の最終行であるステップ１００（０００１００）までのソースコードが抽出、変換されたインターフェースクラス２１０のソースコードである。

図２１、図２２、図２３に示すプログラム中、各位置番号はどのように変換がなされたかを説明するためのものであり、実際のプログラムに記載する必要はない。また、上記において説明されていない位置番号については後述する。

次に、図１６に示す結果出力プログラム１０９からインターフェースクラス２１０への抽出、変換の具体的動作を説明する。図２４、図２５に結果出力プログラム１０９のプログラムを記載する。図２４の第１行であるステップ１（０００００１）から図２５の最終行であるステップ８４（０００００８４）までのソースコードが結果出力プログラム１０９のソースコードである。

上記結果出力プログラム１０９に記載された位置番号（１－２５）から伝票定義を抽出し、図１８の位置番号（１－２５）に示す画面表示メソッドの画面定義に変換する。この場合、伝票の行位置、桁位置を画面の行位置、桁位置に対応付ける必要がある。

また、上記結果出力プログラム１０９の位置番号（１－２５）に示す伝票項目の「ＳＯＵＲＣＥ指定」を図１８の位置番号（１－２

５）に示す画面項目の「ＴＯ指定」に置き換えることが必要となる。

さらに、トランザクションレコードに含まれているがマスタレコードに含まれないデータ項目をもとにそれらの項目について画面項目を生成する。結果出力プログラム１０９はマスタレコードの項目を出力しているが、インターフェースクラス２１０の画面表示メソッドでは画面からトランザクションの項目を入力するためである。

以下に、上記抽出／変換部１６００がマッチングプログラム１０７とソートプログラム１０４からファイルクラス２２０へ変換する情報を抽出する抽出変換処理を図１４を用いて説明する。

・マッチングプログラム１０７とソートプログラム１０４からファイルクラス２２０への抽出変換処理

次に、マッチングプログラム１０７とソートプログラム１０４とに基づいて中間プログラムのファイルクラス２２０を抽出、変換する動作について説明する。前述したように、図１４は左側に示すマッチングプログラム１０７とソートプログラム１０４から必要なデータを抽出し、抽出したデータに基づいてマッチングプログラム１０７またはソートプログラム１０４を変換し、右側に示すファイルクラス２２０を生成するための各対応付けを示している。

マッチングプログラム１０７とソートプログラム１０４とから中間プログラム２００を生成するための各対応についての基本的方針を説明する。

まず、マッチングプログラム１０７からはマッチング更新のロジックを抽出する。また、ソートプログラム１０４からはレコードキーの情報を抽出する。

マッチング更新のロジックは次の３つの対応付けにより変換する

第１はマッチングプログラム１０７ではチェック・ソート済みトランザクションファイル１０５からレコードを読み込んでいたが、ファイルクラス２２０ではトランザクションレコードを手続きの引数として受け取るように変換する必要がある。

第２にマッチングプログラム１０７ではマスタレコードとチェック・ソート済みトランザクションファイル１０５のレコードをマッチングするロジックが存在していたが、ファイルクラス２２０においてもこのロジックが必要となる。

第３にマッチングプログラム１０７ではマッチングした結果、データが正しければ処理区分に応じて、新マスタレコードを更新し、新マスタファイルに書き出す動作を行なっていたが、ファイルクラス２２０ではデータが正しければ処理区分に応じてマスタレコードを更新し、マスタファイルを書き換えるという処理に変換する必要がある。

上記第１～第３の対応付けに従い、抽出／変換部１６００は、マッチングプログラム１０７で存在していたトランザクションファイルからレコードを読み込むロジックを無視し、トランザクションレコードを引数として定義する。

また、抽出／変換部１６００は、マッチングプログラム１０７で存在していたマスタファイルの定義のどちらか一方を無視し（この例では新マスタファイルを無視する）マスタファイルへの読み書きは無視しなかった残る一方のファイルのみに行うようにロジックを変換する。

また、抽出／変換部１６００は、マスタファイルを索引編成ファイルにする。

抽出／変換部１６００は、マスタレコードを書き出す命令を、追加（ＷＲＩＴＥ）、更新（ＲＥＷＲＩＴＥ）、削除（ＤＥＬＥＴＥ）にする。

トランザクションレコード、マスタレコードとも、ファイルクラス２２０においては一件のみ処理するので、抽出／変換部１６００は次のレコードを準備する繰り返しのロジックを無視する。

以上の方針に基づき、抽出／変換部１６００は変換前のマッチングプログラム１０７とソートプログラム１０４とから必要な情報を抽出、変換し、ファイルクラス２２０のソースコードを自動生成する。

上記抽出／変換部１６００による抽出、変換の詳細を図２６～図２８に示す。図２６、図２７は、マッチングプログラム１０７からファイルクラス２２０への抽出、変換の詳細を示す図である。図２８は、ソートプログラム１０４からファイルクラス２２０への抽出、変換を示す図である。図２６～図２８に示された位置番号は図１４の対応付けに示された位置番号と一致している。このように、抽出／変換部１６００が自動的に必要な情報を抽出変換し、ファイルクラス２２０が生成される。

抽出／変換部１６００は、コボルプログラム１００にはないが中間プログラム２００で必要となる要素を新規に生成して追加する。すなわち、インターフェースクラス２１０に次の３つの要素を追加する。第１の要素は、画面からの入力が終了かを判定するフラグ項目である。第２の要素は、画面表示メソッドや、入力チェックメソッドを呼び出すメイン手続である。第３の要素は画面表示メソッド内の表示及び入力受付命令である。

以上に説明したマッチングプログラム１０７のソースコードを図

２９～図３１に示す。また、ソートプログラム１０４のソースコードを図３２に示す。さらに、ファイルクラス２２０のプログラムを図３３、図３４に示す。

プログラム中の位置情報は図１４、図２６及び図２７及び図２８との対応を明確に示すために記載しているが、実際のプログラムには不要である。

本実施の形態の変換装置Ａ１０００によれば、集中型処理に適合したコボルプログラム１００のソースコードから分散型処理に適合した中間プログラム２００のソースコードへの自動変換が可能となる。従って、集中型処理に用いられていたプログラム資産を分散型処理においても使用することができるため、プログラム資産の有効活用が図れる。

また、本発明の実施の形態によれば、旧来コーディングされたプログラム中の業務ロジックを再利用することができる。

また、プログラムの変換作業は変換装置Ａ１０００によって自動的に行われるために、人的労力を必要としないため、労力の低減を図ることができる。

また、企業などの組織において、集中型処理から分散型処理へシステムを移行する場合に、集中型処理に用いていたプログラムを分散型処理においても有効に使用できるため、新たにプログラムを開発する必要を最小限に抑えることができ、人的労力の軽減、システム構築期間の短縮化及びシステム構築費用の低減を図ることができる。

また、手続き型プログラムからオブジェクト指向プログラムに変換することによって、プログラム間のインターフェースをより明確にできる。

また、手続き型プログラムからオブジェクト指向プログラムに変換することによって、オブジェクト内部の仕様変更が外部に及ばないようなプログラムが可能になり、ソースコードを再利用しやすくなる。

次に、図１に記載された中間プログラム２００から最終プログラム３００を生成する変換装置Ｂ２０００について説明する。この変換装置Ｂ２０００により、中間プログラム２００をよりオブジェクト指向的な、個別の役割に特化したクラスからなる最終プログラム３００に変換することができる。

まず、中間プログラム２００から最終プログラム３００へプログラム変換を行う変換装置Ｂ２０００の内部構成について説明する。

図３５は変換装置Ｂ２０００の内部構成図である。

入力部２１００は変換装置Ａ１０００によって変換された中間プログラム２００を入力する。

次に、抽出／変換部２２００が入力部２１００によって入力された中間プログラム２００から必要な情報を抽出し、変換する。出力部２３００は抽出／変換部２２００によって抽出変換されたプログラムを最終プログラム３００として出力する。入力部２１００、抽出／変換部２２００、出力部２３００は必要に応じて記憶部２４００に情報を記憶することができる。たとえば、入力部２１００は入力した中間プログラム２００を記憶部２４００に記憶することができる。なお、記憶部２４００は必ずしも変換装置Ｂ２０００の内部に存在する必要はなく、外部記憶装置を利用してもよい。

次に、変換装置Ｂ２０００により中間プログラム２００を最終プログラム３００に変換する場合において、各データ処理の変換の流れを説明する。

図３６は、変換装置Ｂ２０００によってデータ処理の方法がどのように変換されたかを示す図である。

図の左側は中間プログラム２００によるオンライン一件別処理を示している。図の右側は、最終プログラム３００による、中間プログラム２００よりさらにオブジェクト指向的なオンライン一件別処理を示している。

左側の処理を行う中間プログラム２００はインターフェースクラス２１０とファイルクラス２２０とを持つ。左側のデータの流れについては図３の説明で行なったので、ここでは省略する。

右側の最終プログラム３００は５つのクラスから成り立っている。具体的にはインターフェースクラス２１０からビュークラス３１０，制御クラス３２０及びモデルクラス３３０の３つのクラスが生成される。また、ファイルクラス２２０からセッションクラス３４０及びエンティティクラス３５０の２つのクラスが生成される。

ビュークラス３１０と制御クラス３２０とモデルクラス３３０とはクライアント機器５０００側のクラスである。このうち、ビュークラス３１０は画面表示と入力の受付を行うクラスである。モデルクラス３３０はデータのモデル（属性など）を管理するクラスである。制御クラス３２０はビュークラス３１０によって管理される画面と、モデルクラス３３０によって管理されるデータモデルの制御を行うクラスである。

セッションクラス３４０とエンティティクラス３５０とはサーバ機器６０００側のクラスである。セッションクラス３４０はマスタレコードとトランザクションレコードの照合を行うクラスである。エンティティクラス３５０はセッションクラス３４０によって行なわれた照合の結果を記憶媒体へ書き込むことを管理するクラスであ

る。

このような各クラスの制御に従って、トランザクションレコード３０３をクライアント機器５０００側からサーバ機器６０００側に受け渡し、よりオブジェクト指向的なオンライン一件別処理が可能となる。

次に最終プログラム３００を生成するために抽出／変換部２２００が行う動作について説明する。

まず、抽出／変換部２２００は最終プログラム３００に予め作り込む要素を生成する。生成した要素は、図３６に示した５つのクラスに対応させて５つのオブジェクト指向プログラムのテンプレートに記憶させておく。また、各クラスのファイル名情報を各クラスに対応するテンプレートに付けておく。

また、抽出／変換部２２００は、各クラスにそれぞれ必要なメソッドを作り、中身は空にしておく。この時、意味情報を各メソッドに付与しておき、後で抽出変換の際に各メソッドを特定できるようにする。具体的には、ビュークラス３１０には初期化メソッドと画面表示入力メソッドを作成する。制御クラス３２０には初期化メソッドとＵＩメインメソッドを作成する。モデルクラス３３０には初期化メソッドと入力チェックメソッドと、画面データ受け取りメソッドを作成する。セッションクラス３４０には初期化メソッドとトランザクションチェックメソッドを作成する。エンティティクラス３５０には初期化メソッドとマスタファイル存在チェックメソッドとマッチング更新メソッドとを作成しておく。

次に、抽出／変換部２２００が中間プログラム２００から必要な情報を抽出変換し、最終プログラム３００を生成する動作について、まず、インターフェースクラス２１０からビュークラス３１０、

制御クラス３２０及びモデルクラス３３０を抽出、変換する動作を説明し、その後にファイルクラス２２０からセッションクラス３４０とエンティティクラス３５０とを抽出、変換する動作を説明する。

・インターフェースクラス２１０から３つのクラスへの対応付け

まず、インターフェースクラス２１０から３つのクラスへの対応付けについて説明する。

抽出／変換部２２００は、インターフェースクラス２１０の持つ役割のうち画面表示と入力の役割部分をビュークラス３１０に振り分ける。次に、抽出／変換部２２００は、入力のチェックの役割部分をモデルクラス３３０に振り分ける。そして、抽出／変換部２２００は、これら２つのクラスに振り分けた各役割を呼び出す動作制御を制御クラス３２０に振り分ける。

以上に説明したインターフェースクラス２１０から３つのクラスへの対応付けを図３７に示す。また、各クラスへの具体的な抽出変換方法を図３８～図４０に示す。ここで、図３８～図４０中で示した位置番号は図３７の位置番号に対応している。図３８はインターフェースクラス２１０のプログラムからビュークラス３１０のプログラムへの具体的抽出変換方法を示す。図３９はインターフェースクラス２１０から制御クラス３２０への具体的抽出変換方法を示す。図４０はインターフェースクラス２１０からモデルクラス３３０への具体的抽出変換方法を示す。図３８に示す抽出、変換方法に基づいて抽出／変換部２２００により自動生成されたビュークラス３１０の具体的プログラムのソースコードを図４１、図４２に示す。なお、位置番号は変換前のプログラムのソースコードと変換後のプログラムのソースコードを対応付けるために記載されたものであり

、実際のプログラム中には存在する必要はない。変換前のインターフェースクラス２１０は図２２、図２３に明示されているが、このインターフェースクラス２１０のプログラム中に記載された位置番号（２－１）（２－２）（２－３）と図４１、図４２に示すビュークラス３１０のプログラム中に明示された位置番号（２－１）（２－２）（２－３）とが対応つけられている。

図３９に示す抽出、変換方法に基づいて抽出／変換部２２００によって自動生成された制御クラス３２０のプログラムのソースコードを図４３に示す。インターフェースクラス２１０と制御クラス３２０とは、これら２つのプログラムのソースコードに明示された位置番号（２－１）（２－４）によって対応付けがなされている。

さらに、図４０に示す抽出、変換方法に基づいて抽出／変換部２２００によって自動生成されたモデルクラス３３０のプログラムのソースコードを図４４、図４５に示す。インターフェースクラス２１０とモデルクラス３３０とは、位置番号（２－１）（２－２）（２－５）（２－６）（２－７）（２－８）（２－９）によって対応付けがなされている。

・ファイルクラス２２０から２つのクラスへの対応付け

次に、ファイルクラス２２０から２つのクラス（３４０，３５０）への対応つけについて説明する。抽出／変換部２２００は、ファイルクラス２２０の持つ役割をセッションクラス３４０及びエンティティクラス３５０の２つのクラスに次のように振り分ける。すなわち、ファイルクラス２２０では、マスタファイルの更新又は削除の場合には、トランザクションレコードに該当するものがマスタファイルに存在することを確認し、マスタファイルへの追加の場合には、トランザクションレコードに該当するものがマスタファイルに

存在しないことを確認するロジックが存在していた。抽出／変換部２２００は、これらのロジックをセッションクラス３４０に振り分ける。また、抽出／変換部２２００は、トランザクションレコードの処理区分に応じてマスタファイルにトランザクションレコードを追加または更新または削除するロジックをエンティティクラス３５０に振り分ける。

このような対応づけによりファイルクラス２２０のプログラムのソースコードは変換装置Ｂ２０００によってセッションクラス３４０のプログラムのソースコードとエンティティクラス３５０のプログラムのソースコードに変換される。

上記抽出／変換部２２００によるファイルクラス２２０からセッションクラス３４０及びエンティティクラス３５０への対応付けを図４６に示す。

また、ファイルクラス２２０のプログラムからセッションクラス３４０のプログラムへ変更するための具体的抽出変換方法を図４７に示す。また、ファイルクラス２２０のプログラムからエンティティクラス３５０のプログラムへ変換するための具体的抽出変換方法を図４８に示す。

抽出／変換部２２００は、図４７に示す抽出変換方法に基づいて、ファイルクラス２２０のプログラムから必要なデータを抽出し、抽出したデータを用いてセッションクラス３４０のプログラムに自動変換する。自動変換されたセッションクラス３４０のプログラムを図４９、図５０に示す。

また、抽出／変換部２２００は、図４８に示す抽出変換方法に基づいて、ファイルクラス２２０のプログラムから必要なデータを抽出し、抽出したデータを用いてエンティティクラス３５０のプログ

ラムに自動変換する。自動変換されたエンティティクラス３５０のプログラムを図５１～図５３に示す。

上記生成された最終プログラム３００の５つのクラスには、中間プログラム２００には含まれない要素が存在する。そのため、その要素を新規に生成する必要がある。新規に作成すべき各クラスへの追加の要素を説明する。

この追加の要素作成は、抽出／変換部２２００によって行なわれる。

まず、抽出／変換部２２００がビュークラス３１０への新規事項追加を行う動作について図４１を用いて説明する。

・初期化メソッド

初期化メソッドの内容として自己インスタンスを生成し、制御クラスの初期化メソッドを呼び出すステップを追加する。

具体的には、メソッドのデータ部にＷＯＲＫＩＮＧ－ＳＴＯＲＡＧＥ　ＳＥＣＴＩＯＮを設け、自己インスタンスを参照するステップを追加する（ステップ１９～２０）。メソッドの手続き部には、自己インスタンスを生成するステップ（ステップ２２～ステップ２３）、また自己インスタンスを引数として制御クラスの初期化メソッドを呼び出すステップ（ステップ２４～ステップ２５）を追加する。

次に、抽出／変換部２２００が制御クラス３２０へ追加する事項について図４３を用いて説明する。

・初期化メソッド

初期化メソッドの内容として自己インスタンスを生成し、引数のビューインスタンスと自己インスタンスとをつなげるステップを追加する。またモデルクラス３３０の初期化メソッドを呼び出し、戻

り値のモデルインスタンスを自己インスタンスとつなげるステップを追加する。

具体的には、メソッドのデータ部にＬＩＮＫＡＧＥ　ＳＥＣＴＩＯＮを設け、引数のビューインスタンスを参照するステップを追加する（ステップ２０～ステップ２１）。またＷＯＲＫＩＮＧ－ＳＴＯＲＡＧＥ　ＳＥＣＴＩＯＮを設け、自己インスタンスを参照するステップを追加する（ステップ２２～ステップ２３）。メソッドの手続き部には、ビューインスタンスを引数として受け取ることを宣言するステップ（ステップ２４）、自己インスタンスを生成するステップ（ステップ２５～ステップ２６）、引数のビューインスタンスと自己インスタンスとをつなげるステップ（ステップ２７～ステップ２８）、モデルクラス３３０の初期化メソッドを呼び出し、戻り値のモデルインスタンスを自己インスタンスとつなげるステップ（ステップ２９～ステップ３１）、また制御クラス３２０のＵＩメインメソッドを呼び出すステップ（ステップ３２）を追加する。

次に、抽出／変換部２２００がモデルクラス３３０へ追加する事項について図４４、図４５を用いて説明する。

・初期化メソッド

初期化メソッドの内容として自己インスタンスを生成し、引数のビューインスタンスと自己インスタンスとをつなげ、自己インスタンスを戻り値として設定するステップを追加する。

具体的には、メソッドのデータ部にＬＩＮＫＡＧＥ　ＳＥＣＴＩＯＮを設け、引数のビューインスタンスを参照するステップと戻り値の自己インスタンスを参照するステップを追加する（ステップ２０～ステップ２２）。またＷＯＲＫＩＮＧ－ＳＴＯＲＡＧＥ　ＳＥＣＴＩＯＮを設け、自己インスタンスを参照するステップを追加す

る（ステップ２３～ステップ２４）。メソッドの手続き部には、ビューインスタンスを引数として受取り自己インスタンスを戻り値として返すことを宣言するステップ（ステップ２６～ステップ２７）、自己インスタンスを生成するステップ（ステップ２８～ステップ２９）、引数のビューインスタンスと自己インスタンスとをつなげるステップ（ステップ３０～ステップ３１）、また自己インスタンスを戻り値として設定するステップ（ステップ３２）を追加する。

次に、抽出／変換部２２００がセッションクラス３４０へ追加する事項について図４９、図５０を用いて説明する。

・初期化メソッド

初期化メソッドの内容として自己インスタンスを生成し、エンティティクラス３５０の初期化メソッドを呼び出して戻り値のエンティティインスタンスを自己インスタンスにつなげ、自己インスタンスを戻り値として設定するステップを追加する。

具体的には、メソッドのデータ部にＬＩＮＫＡＧＥ　ＳＥＣＴＩＯＮを設け、戻り値の自己インスタンスを参照するステップを追加する（ステップ１８～ステップ１９）。またＷＯＲＫＩＮＧ－ＳＴＯＲＡＧＥ　ＳＥＣＴＩＯＮを設け、自己インスタンスを参照するステップを追加する（ステップ２０～ステップ２１）。メソッドの手続き部には、自己インスタンスを戻り値として返すことを宣言するステップ（ステップ２２）、自己インスタンスを生成するステップ（ステップ２３～ステップ２４）、エンティティクラスの初期化メソッドを呼び出し、戻り値のエンティティインスタンスを自己インスタンスとつなげるステップ（ステップ２５～ステップ２６）、また自己インスタンスを戻り値として設定するステップ（ステップ２７）を追加する。

・トランザクションチェックメソッド

トランザクションチェックメソッドにはエンティティクラス３５０のマスタレコード存在チェックメソッドを呼び出し、結果が正しければエンティティクラス３５０のマッチング更新メソッドを呼び出すステップを追加する。

具体的には、メソッドのデータ部にＷＯＲＫＩＮＧ－ＳＴＯＲＡＧＥ　ＳＥＣＴＩＯＮを設け、エンティティクラス３５０のマスタレコード存在チェックメソッドからの戻り値を格納するステップを追加する（ステップ６１～ステップ６４）。メソッドの手続き部には、エンティティクラス３５０のマスタレコード存在チェックメソッドを呼び出すステップ（ステップ６７～ステップ６８）、またこの呼び出しの結果が正しければエンティティクラス３５０のマッチング更新メソッドを呼び出すステップ（ステップ６９～ステップ８２）を追加する。

最後に抽出／変換部２２００がエンティティクラス３５０のプログラムへ追加する事項について図５１、図５２を用いて説明する。

・初期化メソッド

初期化メソッドの内容として、自己インスタンスを生成し、これを戻り値として返すステップを追加する。

具体的にはメソッドのデータ部にＬＩＮＫＡＧＥ　ＳＥＣＴＩＯＮを設け、戻り値の自己インスタンスを参照するステップを追加する（ステップ１７～ステップ１８）。またＷＯＲＫＩＮＧ－ＳＴＯＲＡＧＥ　ＳＥＣＴＩＯＮを設け、自己インスタンスを参照するステップを追加する（ステップ１９～ステップ２０）。メソッドの手続き部には、戻り値として自己インスタンスを返すことを宣言するステップ（ステップ２１）、自己インスタンスを生成するステップ

（ステップ２２～ステップ２３）、戻り値に自己インスタンスを設定するステップ（ステップ２４）を追加する。

・マスタレコード存在チェックメソッド

マスタレコード存在チェックメソッドの内容として、レコードキーを引数に受取り、そのキーでマスタファイルを読み込み、その結果をマスタレコード存在フラグに格納して戻り値として返すステップを追加する。

具体的には、メソッドのデータ部にＬＩＮＫＡＧＥ　ＳＥＣＴＩＯＮを設け、引数のレコードキーを格納するステップと戻り値のマスタレコード存在フラグを格納するステップを追加する（ステップ６２～ステップ６６）。メソッドの手続き部には、レコードキーを引数として受取りマスタレコード存在フラグを戻り値として返すことを宣言するステップ（ステップ６７～ステップ６８）、マスタファイルを読み込む準備をするステップ（ステップ７０～ステップ７１）、またマスタファイルを読み込みその結果を戻り値として設定するステップ（ステップ７２～ステップ７７）を追加する。

このようにして変換装置Ｂ２０００は中間プログラム２００のソースコードから最終プログラム３００のソースコードを自動変換することにより、よりオブジェクト指向性の高いプログラムを生成することができる。このようにオブジェクト指向性の高いプログラムに変換することによって、オブジェクト内部の仕様変更が外部に及ばないようなプログラムが可能になり、ソースコードを再利用しやすくなる。よって、過去において作成されたプログラム資産を分散型処理でさらに有効活用することが可能となる。

また、ＷＥＢやＸＭＬなどに容易に連携することができるため、中間プログラム２００よりも現代社会のネットワークシステムによ

り合致したプログラム構造と言える。従って、このようにして、従来のプログラムから自動的に人的労力をかけずに自動変換された最終プログラム３００によれば、現在主流となっている分散型処理のインフラをより有効に活用することができるアプリケーションプログラムとして再利用することが可能となる。

実施の形態２.

次に実施の形態２について説明する。本実施の形態では、実施の形態１のように中間プログラム２００を生成するステップを設けず、コボルプログラム１００のソースコードから直接最終プログラム３００のソースコードを抽出、変換する形態である。

図５４は、本実施の形態の概念図である。

本実施の形態では、変換装置Ｃ３０００がコボルプログラム１００のソースコードを直接最終プログラム３００のソースコードに変換している。このように、変換装置Ｃ３０００が自動的に直接最終プログラム変換を行うことにより、中間プログラム２００を生成する段階を設けずに、集中型処理からＷＥＢやＸＭＬなどに連携できる分散型処理に適したプログラムを短期間に容易に取得することができる。

変換装置Ｃ３０００の構成及び動作について説明する。図５５は、変換装置Ｃ３０００の内部構成図である。

実施の形態１の変換装置Ａ１０００の内部構成を示す図４と比べ、内部構成自体は同一である。ただし、出力部１７００が出力するプログラムが最終プログラム３００であることと、抽出／変換部３１００の動作が一部異なっている。即ち、変換装置Ａ１０００の抽出／変換部１６００では中間プログラム２００を生成するためにデ

ータの抽出変換を行なっていたが、本実施の形態の抽出／変換部３１００では、最終プログラム３００を生成するためにデータの抽出及び変換を行なっている。

入力部１１００はコボルプログラム１００を入力し、記憶部１８００に記憶する。分割部１２００はコボルプログラム１００をまとまりある複数のプログラムに分解し、構文解析部１３００は分解されたそれぞれのプログラムの構文を解析する。プログラム判断部１４００は各プログラムの役割を判断し、節判断部１５００は各プログラム中の節の内容、役割を判断する。

これらの動作を終えた後、抽出／変換部３１００は最終プログラム３００を生成するためのデータの抽出及び変換を行ない、その結果、生成された最終プログラム３００は出力部１７００によって出力される。この場合、記憶部１８００は必要に応じてデータを記憶する領域として利用することができ、また３０００の内部に記憶部１８００が存在しなくても、外部記憶装置に記憶させてもよい。

このように、本実施の形態では、コボルプログラム１００から中間プログラム２００を出力することなく、直接最終プログラム３００を出力するため、変換処理を高速に行え、ＷＥＢやＸＭＬなどに連携できる分散型処理に適したプログラムを短期間に容易に取得することができる。

また、最終プログラム３００を中間プログラム２００と比較すると機能自体は変わらないが、最終プログラム３００ではプログラムの部品化が進むため、既存部品に差分だけを付け加えて必要な動作を実行するプログラムを容易に作成できる。このため、さらに資産価値が高いプログラムを取得することができる。

また、最終プログラム３００を中間プログラム２００をＪａｖａ

（登録商標）言語やＣ＋＋言語等の言語を用いて記述することが可能である。

なお、上記全ての実施の形態では、コボルプログラムを基に変換プログラムを生成したが、変換前のプログラムはコボルプログラムに限る必要はなく、バッチ処理をする構造化されたプログラムであればよい。従って、構造化されたプログラムであれば、上記変換装置により、分散型処理に適合したプログラムに変換することができる。

図５６は、変換装置Ａ、変換装置Ｂ、変換装置Ｃのコンピュータ基本構成図である。

図５６において、プログラムを実行するＣＰＵ（Ｃｅｎｔｒａｌ Ｐｒｏｃｅｓｓｉｎｇ Ｕｎｉｔ）４０は、バス３８を介してモニタ４１、キーボード４２、マウス４３、通信ボード４４、磁気ディスク装置４６等と接続されている。

磁気ディスク装置４６には、オペレーティングシステム（ＯＳ）４７、プログラム群４９、ファイル群５０が記憶されている。ただし、プログラム群４９、ファイル群５０が一体となってオブジェクト指向のプログラム群４９を形成する形態も一実施の形態として考えられる。

プログラム群４９は、ＣＰＵ４０、ＯＳ４７により実行される。

上記各実施の形態では、変換装置Ａ、変換装置Ｂ、変換装置Ｃは、通信ボード４４の機能を使用して、各種ネットワークを経由して接続された機器と通信を行う。

以上に記載した「格納する」、「記憶する」という用語は、記録媒体に保存することを意味する。

すべての実施の形態では、各構成要素の各動作はお互いに関連し

ており、各構成要素の動作は、上記に示された動作の関連を考慮しながら、一連の動作として置き換えることができる。そして、このように置き換えることにより、変換装置の実施形態を変換方法の発明の実施形態とすることができる。

また、上記各構成要素の動作を、各構成要素の処理と置き換えることにより、変換プログラムの実施の形態とすることができる。

また、変換プログラムをコンピュータ読み取り可能な記録媒体に記憶させることで、変換プログラムを記録したコンピュータ読み取り可能な記録媒体の実施の形態とすることができる。

変換プログラムの実施の形態及び変換プログラムを記録したコンピュータ読取り可能な記録媒体の実施の形態は、すべてコンピュータで動作可能なプログラムにより構成することができる。

また、変換プログラムの実施の形態および変換プログラムを記録したコンピュータ読取り可能な記録媒体の実施の形態における各処理はプログラムで実行されるが、このプログラムは、記録装置に記録されていて、記録装置から中央処理装置（ＣＰＵ）に読み込まれ、中央処理装置によって、プログラムに記述された動作が実行されることになる。

また、各実施の形態に記載されたソフトウエアやプログラムは、ＲＯＭ（ＲＥＡＤ　ＯＮＬＹ　ＭＥＭＯＲＹ）に記憶されたファームウエアで実現されていても構わない。あるいは、ソフトウエアとファームウエアとハードウエアとの組み合わせで前述したプログラムの各機能を実現しても構わない。

実施例.

以下に、一括処理方式のコボルプログラム（レガシープログラム

）の変換法の実務プログラムへの適用実施例について説明する。この実施例では、前述した実施の形態と同一又は相当する部分に対しては、前述した実施の形態と同一符号を用いて説明する。

前述した実施の形態では、一括処理方式のＣＯＢＯＬのレガシープログラムを、クライアント／サーバやＷｅｂ技術を利用した新しい形のプログラムに自動変換する手法を説明した。すなわち、前述した実施の形態では、一括処理を行うＣＯＢＯＬプログラムを対象とし、一件別処理のオブジェクト指向ＣＯＢＯＬプログラムに自動的に変換するアルゴリズムを説明した。これを実際に企業で使用されているＣＯＢＯＬプログラムに適用し、前述した実施の形態で説明した変換法がある程度有効であることを確かめたので、以下に、この実施例の変換法と、実験に用いたＣＯＢＯＬプログラムの内容、実験結果について述べる。

１．はじめに

本実験の目的は、集中型、オフライン一括処理を行う手続き型のプログラムを、オンライン一件別のオブジェクト指向プログラムに自動的に変換し、さらにクライアント／サーバ環境で動作するＷｅｂプログラムに自動的に変換する方法の確立である（図５７）。

この実験例では、一括処理のＣＯＢＯＬプログラムソースコードを元に、クライアント側とサーバ側に分かれ、クライアント側ではＭＶＣ（Ｍｏｄｅｌ，Ｖｉｅｗ，Ｃｏｎｔｒｏｌｌｅｒ）パターンに対応し、サーバ側ではＥＪＢ（ＥｎｔｅｒｐｒｉｚｅＪａｖａＢｅａｎｓ（商標））のトランザクション処理モデルに対応した構造のオブジェクト指向ＣＯＢＯＬ（ＯＯ－ＣＯＢＯＬ）プログラムソースコードを生成する方法の実装を試みた。

この実験例では、変換の方針、規則などを検討するにあたっては

、ＣＯＢＯＬシステムの資料（日立製作所：ＣＯＢＯＬ８５プログラミング、６１９０－３－７２４（１９９５））に載っているサンプルのプログラムを使用した。そして、この変換方法が企業で実際に使用されているＣＯＢＯＬプログラムに対して適用できるか確かめることにした。今回システムインテグレータ１社の協力によりその機会を得た。以下、変換のアルゴリズムと実務プログラムを使用した実験について述べる。

２．サンプルを元にしたプログラム変換のアルゴリズム

プログラムの変換は、２段階に分けて行う（図５８、図５９）。まず、意味付与などの前処理をした後、手続きの流れを一括処理から一件別処理に変えながらクライアント側とサーバ側の２種類のクラス・プログラム２１０、２２０に変換する（図５８）。この変換プログラムを、ここでは、変換１プログラムと呼ぶことにする（あるいは、単に、変換１ともいう）。また、変換１プログラムを実行するハードウエアとソフトウエアの環境を第１の変換部という。

次いでこの２種類のクラス・プログラム２１０、２２０を、クライアント側はＭＶＣの３種類のクラス・プログラム３１０、３２０、３３０に変換し、またサーバ側はＳｅｓｓｉｏｎとＥｎｔｉｔｙの２種類のクラス・プログラム３４０、３５０に変換する。この変換プログラムを、ここでは、変換２プログラムと呼ぶことにする（あるいは、単に、変換２ともいう）。また、変換２プログラムを実行するハードウエアとソフトウエアの環境を第２の変換部という。第１の変換部と第２の変換部とは同一の計算機で実現できる。

（２－１）変換１プログラムによるプログラムへの意味の自動付与

意味の付与とは、プログラムの役割や、プログラム内のデータの宣言・ひと続きの手続き等の要素が一連の処理の中で果たす役割を

検出し、プログラムやその要素に対して役割を表すラベルを付与することである。

意味の付与は、次のことを行うための準備として必要である。

（１）ＣＯＢＯＬプログラムをフレームワークに対応づけて変換するための、プログラムの役割の判断とそのデータや手続きの役割の判断。

（２）プログラムの手続きを一括処理から一件別処理に変えるための、繰り返しなど一括処理のロジックの検出。

予め個々のプログラムに意味のラベルを付与し、さらに、データや手続きに意味のラベルを付与しておき、変換を行う際に要素を抽出するための手がかりとする。

プログラムの役割判断やデータの役割判断、特定のロジックなどの要素の判定は、構文のパターンだけでは判別できないため、意味の付与によって補う。

この意味の付与は、プログラムソースコード内にあるデータの定義を参照し、マスタファイルの定義であるとか、トランザクションファイルの定義であるとかを自動的に判定し、判定結果をプログラムソースコード内に自動で付与する前処理プログラムルーチンにより実現できる。この意味の付与は、変換１プログラムの前段処理で実行する。

（２－２）変換１プログラムによる変換

変換１プログラムでは、図５９に示した４種類のＣＯＢＯＬプログラム１０２、１０４、１０７、１０９を、クライアント側とサーバ側の２種類のＯＯ－ＣＯＢＯＬクラス２１０、２２０に変換する。クライアント側クラス２１０は、画面の入出力、入力データのチェックなど、ユーザ・インタフェースの機能を持つ。またサーバ側

クラス２２０は、トランザクションのマッチング・更新の機能を持つ。

変換では、予め２つのクラスの構造と、必要なメソッドのスケルトンを持つテンプレートを用意し、そこにＣＯＢＯＬプログラムから情報を抽出・変換してあてはめる方式を採る。

ａ．クライアント側クラス２１０の生成

クライアント側クラス２１０は、画面などから入力データを受け付けてトランザクション・レコードに格納し、チェック処理を行い、正しいデータはチェック済トランザクションに格納してサーバ側クラス２２０に渡す。この処理を行う属性とメソッドをクラス内に生成するために、入力・チェックプログラム１０２および結果出力プログラム１０９から要素の抽出・変換を行う（図６０）。

ｂ．サーバ側クラス２２０の生成

サーバ側クラス２２０は、クライアント側クラス２１０からトランザクション・レコードを受け取り、マスタ・レコードとのマッチングを行った結果が正しければ、追加、更新、削除などの処理を行う。ソートプログラム１０４およびマッチング・更新プログラム１０７から要素の抽出・変換を行う（図６１）。

（２－３）変換２プログラム

変換２プログラムでは、変換１プログラムによって生成したクライアント側クラス２１０をＭＶＣパターンに対応付けて３種類のクラス３１０、３２０、３３０に分け、サーバ側クラス２２０をＥＪＢモデルに対応付けて２種類のクラス３４０、３５０に分ける（図６２、図６３）。

変換１プログラムと同様に、変換２プログラムは予め５つのクラスの骨格を持つテンプレートを用意しておき、そこに変換１プログ

ラムによるＯＯ－ＣＯＢＯＬクラス２１０、２２０から情報を抽出・変換してあてはめる。

ａ．クライアント側クラス２１０のＭＶＣクラスへの分解

クライアント側クラス２１０で行っていた処理のうち、画面表示処理と入力の受付け処理をＶｉｅｗクラスに、チェック処理をＭｏｄｅｌクラスに、これら２つのクラスの制御の役割をＣｏｎｔｒｏｌｌｅｒクラスにふり分ける（図６２）。

ｂ．サーバ側クラス２２０のＳｅｓｓｉｏｎクラス３４０、Ｅｎｔｉｔｙクラスへ３５０の分解

サーバ側クラス２２０で行っていた処理のうち、トランザクション・レコードとマスタ・レコードとのマッチング処理をＳｅｓｓｉｏｎクラス３４０に、追加、更新、削除などの更新処理をＥｎｔｉｔｙクラス３５０にふり分ける（図６３）。

以上がこの実施例の基礎となる基本的変換法である。この変換法について、ＣＯＢＯＬを扱うシステムインテグレータ１社で実際に運用されているプログラムを使用して評価を行う機会を得た。実験内容について以下の「３．実務プログラムへの適用実験」で述べる。

３．実務プログラムへの適用実験

３つのジョブ、計１７本のプログラムの入出力や手続きの特徴を分析し、次の３種類の一括処理に分けられることがわかった。

（種類１）トランザクションを入力し、キーの変わり目を判定して集計を出力する処理。

（種類２）トランザクションを入力し、新しい項目を追加するなどレコードの形式を変えて出力する処理。

（種類３）うえの２種類にあてはまらない処理。

上記計１７本のプログラムの中から上記（１）、（２）、（３）にそれぞれに該当する典型的なプログラムを各１本、計３つの一括処理プログラムを抽出した。以下に実験の方針、個々のプログラムの変換の実際について述べる。

（３－１）実験の方針

実験の目的と手順を次のように設定して行った。

（１）実験の目的

サンプルのプログラムを用いて、変換アルゴリズムが、実務のプログラムにもあてはまるか検証する。すなわち、

（検証目的１）：命名規則からプログラム中のデータの役割が判断できるか。

（検証目的２）：手続きの節、段落の呼出し関係から、個々の要素の役割を導出判断できるか。

（検証目的３）：データの定義を移動する操作、繰返しのロジックを除去する操作、などの要素に対する操作を行えるか。

主にこれらの前提が実務のプログラムにもあてはまるか、変換を試行して確かめる。

（２）実験の手順

ａ．変換の方針の決定

最終的にどのようなＯＯ－ＣＯＢＯＬプログラムに変換するか検討する。

①データの入出力の変更、

②手続きの一括処理から一件別処理への変更、

③必要に応じて、元のプログラムのデータを組み合わせた新しいファイルの導入、

などの変換の方針を決める。

ｂ．変換後のプログラムの構築

決定した方針に従って、変換して出来上がったＯＯ－ＣＯＢＯＬプログラムを人手で組む。

ｃ．変換規則の検討

元のプログラムと、上記「ｂ．変換後のプログラムの構築」で構築した変換後のプログラムとを比較し、元のプログラム内の要素に対する変換の規則を検討する。

上記「ａ．変換の方針の決定」、「ｂ．変換後のプログラムの構築」、「ｃ．変換規則の検討」は、変換１プログラムと変換２プログラムとを作成するために行うものであり、変換１プログラムと変換２プログラムが一旦作成された場合は、実行する必要はない。ただし、上記「ａ．変換の方針の決定」、「ｂ．変換後のプログラムの構築」、「ｃ．変換規則の検討」は、３つの一括処理プログラムを対象にそれぞれ個々に行うものであり、３つの一括処理プログラムそれぞれに対して、変換１プログラムと変換２プログラムとがそれぞれ３種類作成されることになる。３種類の変換１プログラムと変換２プログラムが一旦作成された後は、その種類に属する集中型、オフライン一括処理を行う手続き型のプログラムソースコードとデータの命名規則と手続きの組み方の規則を入力データとして変換１プログラムを実行することにより、オンライン一件別のオブジェクト指向プログラムに自動的に変換でき、さらに、変換２プログラムを実行することにより、クライアント／サーバ環境で動作するＷｅｂプログラムに自動的に変換できる。

変換のプロセス中で最も複雑な操作を行っているのは、変換１プログラムである。変換１は、元のプログラムの入出力を変更し手続きを一括処理から一件別処理に変更する段階である。そこでこの変

換１の段階について特に詳細に述べる。

（３－２）一件別処理への変更

データと手続きの特徴に基づき以下の３つの一括処理プログラムを抽出した。

（１）第１の一括処理プログラム：「集計・出力プログラム９０１」

（２）第２の一括処理プログラム：「レコード形式変換プログラム９０２」

（３）第３の一括処理プログラム：「出力指示データ作成プログラム９０３」

抽出した３つの一括処理プログラムを、次のように一件別処理に変更し、一つのメソッドとして利用する。以下に、３つの一括処理プログラムの各プログラムについて、

ａ．現行の処理内容、

ｂ．変換後の処理内容、

ｃ．手続きの変換、

ｄ．それらに基づく変換規則

を述べる。

（１）第１の一括処理プログラム「集計・出力プログラム９０１」

ａ．現行の処理内容

トランザクションをファイルから順次読み込み、部門、店の変更（ブレイク）があれば、小計合計を計算・出力する一括処理を行う（図６４）。このときマスタＤＢから文字情報を取得して印字する。

ｂ．変更後の処理内容

トランザクションをレコードとして一件別に受け取り、一件ごと

に随時に小計合計を計算する。計算結果は新しく作る集計ファイル（索引編成）に累積していく。照会するときは、この集計ファイルからデータを出力する（図６５）。

ｃ．手続きの変換

現行の一括処理の手続きの流れは、次のとおりである（図６６）。

①トランザクションファイルから１レコード読む。

②明細の数値を各小計に加算する。

③小ブレイクならば、小計の合計を計算し、各小計と共に出力する。

④大ブレイクならば、全合計を計算し、各合計と共に出力する。

⑤トランザクションファイルが終わるまで繰り返す。

この手続きを、上記「ｂ．変更後の処理内容」の方針に従って次の一件別の流れに変換する（図６７）。

①１レコードが引数として入力される。

②明細の数値を各小計に加算する。

③小計の合計を計算し、集計ファイルの該当する小計レコードを更新する。

④全合計を計算し、集計ファイルの該当する合計レコードを更新する。

ｄ．変換規則の検討

プログラムに以下のような変換を施す。

①一連の手続きをループする繰返しの指定を除去する。

②トランザクション・レコードの定義を、引数にするためにデータ部連絡節に移動する。

③ブレイクの判定を除去し、毎回ブレイク処理を行うようにする（

図６８）。すなわち、ＩＦクローズによる、部門、店の変更（ブレイク）判定を削除して、トランザクション・レコード１件ごとに、小計合計を計算するＰＥＲＦＯＲＭ文を無条件に実行する。

④リスト出力の部分を、集計ファイルの該当レコードの読込み、集計項目の加算、ファイルの更新、に変える。

（２）第２の一括処理プログラム「レコード形式変換プログラム９０２」

ａ．現行の処理内容

トランザクションをファイルから順次読み込み、他のレコード形式に変換してファイルに書き出す一括処理を行う。

ｂ．変更後の処理内容

トランザクションを一件別にレコードの形式で受け取り、他の形式のレコードに変換し、次の処理への引数として送り出す（図６９）。

ｃ．手続きの変換

現行の一括処理の手続きの流れは、次のとおりである（図７０の「（１）現行」）。

①トランザクションファイルから１レコード読む。

②レコードの各項目の値を、新しいレコードに転記する。

③新しいレコードを新ファイルに書き出す。

④トランザクションファイルが終わるまで繰り返す。

この手続きを、上記「ｂ．変更後の処理内容」の方針に従って次の一件別の流れに変換する（図７０の「（２）変換後」）。

①１レコードが引数として入力される。

②レコードの各項目の値を、新しいレコードに転記する。

③新しいレコードを次の処理を行うメソッドの引数として送り出す

d．変換規則の検討

プログラムに以下のような変換を施す。

①一連の手続きをループする繰返しの指定を除去する（図７１）。すなわち、ＵＮＴＩＬクローズによるループ処理を削除し、ＰＥＲＦＯＲＭ文のみにする。

②トランザクション・レコードの定義を、引数にするためにデータ部連絡節に移動する。

③新しいレコードを書き出す部分を、次の処理を行うメソッドの呼出しに変える。

（３）第３の一括処理プログラム「出力指示データ作成プログラム９０３」

①．現行の処理内容

トランザクションをファイルから順次読み込み、同じＩＤに属する下位のＩＤを並べて出力指示データとして書き出す一括処理を行う（図７２）。

②．変更後の処理内容

トランザクションを一件別にレコードの形式で受け取る。出力指示データは索引ファイルに格納し、トランザクションごとに随時更新を行う（図７３）。

③．手続きの変換

現行の一括処理の手続きの流れは、次のとおりである（図７４の「（１）現行」）。

①トランザクションファイルから１レコード読む。

②ソートファイルに書き出す。

③全て書き出した後、ソートを行う。

④ソートファイルから１レコード読む。

⑤出力レコード内の配列構造に指示データを転記する。

⑥出力レコードを書き出す。

⑦ファイルが終わるまで繰り返す。

この手続きを、上記「ｂ．変更後の処理内容」の方針に従って次の一件別の流れに変換する（図７４の「（２）変換後」）。

①１レコードが引数として入力される。

②出力レコード内の配列構造に指示データを転記する。

③出力指示データのファイルから、該当するレコードを更新する。

ｄ．変換規則の検討

プログラムに以下のような変換を施す。

①ソート用レコードの定義を、引数にするためにデータ部連絡節に移動する。

②ソートの命令文からソート前手続きとソート後手続きの呼び出し指定を抽出し、ＰＥＲＦＯＲＭ文による呼び出しに置き換えるほか、ソートに関する命令を除去する。

③出力レコードを書き出す部分を、出力指示データのファイルに対する更新に置き換える。

以上のように、本変換法を適用した変換の方針の決定、手続きの変換、変換規則の決定を行うことができた。

（３－３）クライアント／サーバ（Ｃ／Ｓ）のクラスへの分解

一括処理のプログラム９０１から一件別処理のメソッド９０１ａの形に変換したプロダクトのうち、計算、転記を行ってデータの内容を変更する部分をクライアント側クラス２１０に分け、最終的な結果をファイルに読み書きする部分をサーバ側クラス２２０に分ける（図６７から図７５へ変換する）。一件別のメソッドへの変換結

果を元にして、クライアント／サーバ（Ｃ／Ｓ）のクラスへの分解を自動的に行う。サーバ側クラス２２０の読み書きのロジックは新規のものなので、テンプレートにデータ名等をあてはめてＣ／Ｓのクラスを生成する。

（３－４）Ｃ／Ｓのクラスから５つのクラスへの分解

クライアント側クラス２１０にあったロジックのほとんどはＭｏｄｅｌクラス３３０に、サーバ側クラス２２０にあったロジックはＥｎｔｉｔｙクラス３５０に入る。他のクラスは、データをそのまま受け渡ししたり、メソッドの呼び出し制御を行う（図７５から図７６へ変換する）。Ｃ／Ｓのクラスへの分解と同様に、Ｃ／Ｓのクラスから自動的に５つのクラス３１０、３２０、３３０、３４０、３５０を生成する。また５つのクラスを生成する際にテンプレートを用いる点もＣ／Ｓのクラスの生成と同様である。

４．考察

ａ．アルゴリズムの実用性

実験の目的にあげた、

（検証目的１）：プログラム中のデータの役割の判断、

（検証目的２）：手続きの節、手続きの呼出し関係からの個々の要素の役割の導出判断、

（検証目的３）：プログラム中の要素に対する操作、

などが行え、本変換法が実務のプログラムにも適用できることを確かめた。企業ではコーディング規約を定めてプログラミングを行っていることが多く、同種の複数のプログラムに対しこれらのアルゴリズムを適用できる可能性はある。

一方、変換を行う前にまず人手で変換の方針を検討しなければならず、その手間がかかる点が本変換法の課題である。

b．変換による量的な変化

変換のプロセスで、元のプログラムの内容が最も変化するのが、一括処理のプログラムから一件別のメソッドの形式に変える段階である。図７７に、この段階で変化する行数がプログラムに占める割合を示す。形式変換プログラムの変化の割合が大きいのは、処理が単純で行数が少ないのに対し、項目数の多いレコードを移動していることが原因である。

また、元のプログラムソースコードからＣ／Ｓクラスに変換すると、プログラムソースコードの行数は元のプログラムの約１．４倍になり、最終的な５つのクラスのプログラムソースコードの行数は元のプログラムの１．６倍になる。

５．おわりに

以上のように、この実施例では、レガシープログラムの再利用を目的とした変換法について、実務で適用できる可能性があることを示した。

産業上の利用可能性

本発明の実施の形態によれば、旧来のプログラム資産を分散システムに対応するプログラム資産に変換することができる。

また、本発明の実施の形態によれば、旧来コーディングされたプログラム中の業務ロジックを再利用することができる。

また、本発明の実施の形態によれば、旧来のプログラム資産を利用して、新しいシステムを構築することができる。

また、本発明の実施の形態によれば、旧来のプログラム資産から中間プログラムへ変換することなく直接最終プログラムへ変換することができる。

請求の範囲

1．バッチ処理をするプログラムをソースコードの形式で記憶する記憶部と、

上記記憶部が記憶したプログラムのソースコードを１以上のまとまりある処理に区切り、区切った処理を節として節毎の役割を各節の意味情報として判断する節判断部と、

上記節判断部が判断した各節の意味情報に基づいて上記記憶部が記憶したプログラムのソースコードからソースコード変換のための変換情報を抽出し、抽出した変換情報に基づいてプログラムのソースコードをクライアント機器用の変換結果プログラムのソースコードとサーバ機器用の変換結果プログラムのソースコードとの２つからなる変換結果プログラムのソースコードに変換する抽出／変換部とを備える変換装置。

2．上記抽出／変換部は、上記２つの変換結果プログラムのソースコードを、オブジェクト指向プログラムのソースコードに変換する請求項１に記載された変換装置。

3．上記抽出／変換部は、所定のデータ構造と手続きをもつ複数のクラスに対応した複数のオブジェクト指向プログラムのテンプレートを生成し、上記２つの変換結果プログラムのソースコードから所定のデータ構造と手続きからなる情報を複数抽出し、抽出した各情報をテンプレートの対応する部分に適用することによって上記２つの変換結果プログラムのソースコードを複数のオブジェクト指向プログラムのソースコードに変換する請求項２に記載された変換装置。

4．バッチ処理をするプログラムをソースコードの形式

で記憶する記憶部と、

上記記憶部が記憶したプログラムのソースコードを１以上のまとまりある処理に区切り、区切った処理を節として節毎の役割を各節の意味情報として判断する節判断部と、

所定のデータ構造と手続きをもつ複数のクラスに対応した複数のオブジェクト指向プログラムのテンプレートを生成し、上記節判断部が判断した各節の意味情報に基づいて上記記憶部に記憶されたバッチ処理をするプログラムのソースコードから所定のデータ構造と手続きからなる情報を複数抽出し、抽出した各情報をテンプレートの対応する部分に適用することによって上記記憶部が記憶したプログラムのソースコードを複数のオブジェクト指向プログラムのソースコードに変換する変換装置。

５．上記変換装置は、さらに、

上記記憶部が記憶したプログラムのソースコードの役割をプログラムの意味情報として判断するプログラム判断部を備え、

上記抽出／変換部は、上記プログラム判断部が判断したプログラムの意味情報と上記節判断部が判断した各節の意味情報とに基づいてプログラムのソースコードからプログラムのソースコードを変換するための変換情報を抽出する請求項１に記載された変換装置。

６．上記変換装置は、さらに、

上記記憶部が記憶したプログラムを構文解析する構文解析部を備え、

上記節判断部は、上記構文解析部によって構文解析されたプログラムに含まれる各節の意味情報を判断する請求項１に記載された変換装置。

７．上記変換装置は、バッチ処理をするコボルプログラ

ムのソースコードを変換する請求項１に記載された変換装置。

８．バッチ処理をするプログラムをソースコードの形式で記憶し、

上記記憶したプログラムのソースコードを１以上のまとまりある処理に区切り、区切った処理を節として節毎の役割を各節の意味情報として判断し、

上記判断した各節の意味情報に基づいて上記記憶したプログラムのソースコードからソースコード変換のための変換情報を抽出し、抽出した変換情報に基づいてプログラムのソースコードをクライアント機器用の変換結果プログラムのソースコードとサーバ機器用の変換結果プログラムのソースコードとの２つからなる変換結果プログラムのソースコードに変換する変換方法。

９．バッチ処理をするプログラムをソースコードの形式で記憶する処理と、

上記記憶したプログラムのソースコードを１以上のまとまりある処理に区切り、区切った処理を節として節毎の役割を各節の意味情報として判断する処理と、

上記判断した各節の意味情報に基づいて上記記憶したプログラムのソースコードからソースコード変換のための変換情報を抽出し、抽出した変換情報に基づいてプログラムのソースコードをクライアント機器用の変換結果プログラムのソースコードとサーバ機器用の変換結果プログラムのソースコードとの２つからなる変換結果プログラムのソースコードに変換する処理とをコンピュータに実行させる変換プログラム。

１０．バッチ処理をするプログラムをソースコードの形式で記憶する処理と、

上記記憶したプログラムのソースコードを１以上のまとまりある処理に区切り、区切った処理を節として節毎の役割を各節の意味情報として判断する処理と、

上記判断した各節の意味情報に基づいて上記記憶したプログラムのソースコードからソースコード変換のための変換情報を抽出し、抽出した変換情報に基づいてプログラムのソースコードをクライアント機器用の変換結果プログラムのソースコードとサーバ機器用の変換結果プログラムのソースコードとの２つからなる変換結果プログラムのソースコードに変換する処理とをコンピュータに実行させるための変換プログラムを記録したコンピュータ読み取り可能な記録媒体。

１１．オフライン一括処理を行う手続き型のプログラムソースコードを入力データとして、このプログラムソースコードをオンライン一件別処理のプログラムに変換する第１の変換部と、このオンライン一件別処理のプログラムをクライアント／サーバ環境で動作するＷｅｂプログラムに変換する第２の変換部とを備えたことを特徴とする変換装置。

１２．上記第１の変換部は、オフライン一括処理を行うプログラムソースコードとデータの命名規則と手続きの組み方の規則を入力データとして、上記プログラムソースコードをオンライン一件別処理を行うクライアント側クラスとサーバ側クラスの２種類のクラス・プログラムに変換し、

上記第２の変換部は、この２種類のクラス・プログラムを入力し、オブジェクト指向プログラムのソースコードを生成することを特徴とする請求項１１記載の変換装置。

１３．上記第２の変換部は、クライアント側クラスをモデ

ルクラスとビュークラスとコントローラクラスとの３種類のクラス・プログラムに変換し、またサーバ側クラスをセッションクラスとエンティティクラスとの２種類のクラス・プログラムに変換することを特徴とする請求項１２記載の変換装置。

１４．上記第１の変換部は、上記プログラムのソースコード内にあるデータの定義を参照し、マスタファイルの定義とトランザクションファイルの定義とを判定して、プログラムの役割とそのプログラム内の要素が一連の処理の中で果たす役割とを検出し、プログラムの役割とその要素が一連の処理の中で果たす役割を表すラベルを付与する意味付与の前処理を備えたことを特徴とする請求項１２記載の変換装置。

１５．上記第１の変換部は変換１プログラムを実行し、上記第２の変換部は変換２プログラムを実行し、

オフライン一括処理を行うプログラムは種類分けされ、

上記変換１プログラムと上記変換２プログラムは、オフライン一括処理を行うプログラムの各種類ごとに対応して作成され、

上記第１の変換部と上記第２の変換部とは、入力したオフライン一括処理を行うプログラムの種類に対応して作成された変換１プログラムと変換２プログラムをそれぞれ実行することを特徴とする請求項１２記載の変換装置。

１６．オフライン一括処理を行う手続き型のプログラムソースコードを入力し、

このプログラムソースコードをオンライン一件別処理のプログラムに変換し、

このオンライン一件別処理のプログラムをクライアント／サーバ環境で動作するＷｅｂプログラムに変換することを特徴とする変換

方法。

要　約　書

本発明は、既存のプログラムを新しい技術のソフトウェアに適した構造に変換することを目的とする。入力部１１００はコボルプログラム１００を入力し、記憶部１８００に記憶する。分割部１２００はコボルプログラム１００をまとまりある複数のプログラムに分解し、構文解析部１３００は分解されたそれぞれのプログラムの構文を解析する。プログラム判断部１４００は各プログラムの役割を判断し、節判断部１５００は各プログラム中の節の内容、役割を判断する。これらの動作を終えた後、抽出／変換部３１００は最終プログラム３００を生成するためのデータの抽出及び変換を行ない、その結果、コボルプログラム１００のソースコードから最終プログラム３００のソースコードへの変換が自動的に行われる変換装置を提供する。